

A Project Report on
Machine Learning Techniques for Accurate Flight Delay Forecasting
submitted in the partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE AND ENGINEERING
By

Devangam Harathi	212G1A0571
Shaik Sohail	222G5A0514
Madiga Shireesha	212G1A0594
K. Kiran Kumar	212G1A0591
V. Adi Keshava Reddy	212G1A05C1
V. Pradeep Kumar Naik	212G1A05B9

Under Supervision of
Dr. K. Bhargavi M. Tech, Ph.D.
Associate Professor & HOD Department of Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES
Approved by AICTE, New Delhi & Affiliated to J.N.T.U. Ananthapuram, Accredited by NAAC
Near S.K. University, Itikalapalli(v), Anantapur (Dt) – 515721A.P.
(2021-2025)

ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES

Approved by AICTE, New Delhi & Affiliated to J.N.T.U. Ananthapuram, Accredited by NAAC
Near S.K. University, Itikalapalli(v), Anantapur (Dt) – 515721.A.P.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the “**Machine Learning Techniques for Accurate Flight Delay Forecasting**” is bonafide work carried out by following students of this institute under guidance of **Dr. K. BHARGAVI** M. Tech, Ph.D., Associate Professor & HOD, Department of CSE, for the partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** from ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES, Itikalapalli(v), Anantapur in the academic year of 2024 - 2025.

Devangam Harathi	212G1A0571
Shaik Sohail	222G5A0514
Madiga Shireesha	212G1A0594
K. Kiran Kumar	212G1A0591
V. Adi Keshava Reddy	212G1A05C1
V. Pradeep Kumar Naik	212G1A05B9

Name of the Supervisor

Dr. K. Bhargavi M. Tech, Ph.D.,
Associate Professor & HOD,
Department of CSE,
Anantha Lakshmi Institute of Technology
and Sciences, Anantapur.

Head of the Department

Dr. K. Bhargavi M. Tech, Ph.D.,
Associate Professor & HOD,
Department of CSE,
Anantha Lakshmi Institute of Technology
and Sciences, Anantapur.

Viva Voce Conducted on: _____

Internal Examiner

External Examiner

DECLARATION

We hereby declare that the work which is being presented in this dissertation entitled “**Machine Learning Techniques for Accurate Flight Delay Forecasting**” submitted towards the partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING, ANANTHA LAKSHMI INSTITUTE OF TECHNOLOGY AND SCIENCES** is an authentic work carried out by us during 2024-2025 under the supervision of **Dr. K. Bhargavi M. Tech., Ph.D., Associate Professor & HOD, Department of Computer Science and Engineering**, Anantha Lakshmi Institute of Technology and Sciences, Itikalapalli(v), Anantapur.

The matter embodied in this dissertation report has not been submitted by us for the award of any other degree or diploma. Further, the technical details furnished in the various chapters in this thesis are purely relevant to the above project.

With Gratitude

Devangam Harathi	212G1A0571
Shaik Sohail	222G5A0514
Madiga Shireesha	212G1A0594
K. Kiran Kumar	212G1A0591
V. Adi Keshava Reddy	212G1A05C1
V. Pradeep Kumar Naik	212G1A05B9

ACKNOWLEDGEMENTS

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we have now the opportunity to express our gratitude for all of them.

It is with immense pleasure that we would like to express my indebted gratitude to **Dr. K. Bhargavi** M. Tech, Ph. D., **Associate Professor & HOD, DEPARTMENT OF CSE, Anantha Lakshmi Institute of Technology and Sciences**. Who has guided us a lot and encouraged us in every step. We thank her for the stimulating guidance, constant encouragement and constructive criticism which have made possible to bring out this project work.

We wish to convey our special thanks to **DR. B.M.G. PRASAD** M. Tech, Ph.D., **DEAN of CSE DEPARTMENT, Anantha Lakshmi Institute of Technology and Sciences**, for giving the required information in doing my project work.

We wish to convey our special thanks to **Dr. Ramamurthy** M. Tech, Ph.D., **Principal of Anantha Lakshmi Institute of Technology and Sciences** for giving the required information in doing the project work. Not to forget, we thank all other faculty and non-teaching staff, and friends who had directly or indirectly helped and supported us in completing the project in time.

We also express our sincere thanks to **Sri M. Anantha Ramudu**, **Chairman** and **Sri M. Ramesh Naidu**, **Vice Chairman** of **Anantha Lakshmi Institute of Technology and Sciences** for providing excellent facilities.

With Gratitude

Devangam Harathi	212G1A0571
Shaik Sohail	222G5A0514
Madiga Shireesha	212G1A0594
K. Kiran Kumar	212G1A0591
V. Adi Keshava Reddy	212G1A05C1
V. Pradeep Kumar Naik	212G1A05B9

TABLE OF CONTENTS

Contents	Page. No
LIST OF FIGURES	viii
LIST OF ABBREVIATIONS	ix
ABSTRACT	x
CHAPTER 1: INTRODUCTION	1- 3
1.1 Overview	1
1.2 Predictive Modelling for Flight Delays	1
1.2.1 Overview of Statistical and Machine Learning Approaches	1
1.3 Machine Learning Techniques for Flight Delay Prediction	1
1.3.1 Classification Models Used in this Study	2
1.4 Comparative Analysis of Classification Models	2
1.4.1 Logistic Regression	2
1.4.2 Random Forest	2
1.4.3 Gradient Boosting	3
1.5 Study Context and Data Collection	3
1.5.1 Dataset Overview	3
CHAPTER 2: LITERATURE SURVEY	4 -7
2.1 Flight Delay Prediction Systems (2020)	4
2.2 Data Mining Applications in Civil Activation Sector: State-of-the-Art Review	4
2.3 Mining Aviation Data to Understand Impacts of severe Weather	5
2.4 Predicting Ground Delay Program at an Airport Based on Meteorological Conditions	5
2.5 A Novel Approach: Airline Delay Prediction Using Machine Learning.	5
2.6 A Machine Learning Approach to Predict Flight Delays	6
2.7 Predicting Airline Delays Using Data Analytics	6
2.8 Impact of Weather on Flight Delays: A Statistical Approach	6

2.9 Deep Learning for Flight Delay Forecasting: A Case Study	7
2.10 Flight Delay Prediction Using Big Data Analytics	7
CHAPTER 3: SYSTEM ANALYSIS	8-11
3.1 System Study	8
3.1.1 Economical Feasibility	8
3.1.2 Technical Feasibility	8
3.1.3 Social Feasibility	9
3.2 Existing System	9
3.3 Disadvantages of Existing System	10
3.4 Proposed System	10
3.5 Advantages of Proposed System	10
3.6 System Requirements	11
3.6.1 Hardware Requirements	11
3.6.2 Software Requirements	11
CHAPTER 4: PROPOSED SYSTEM	12-19
4.1 Proposed System	12
4.1.1 Objectives	12
4.1.2 Advantages of the Proposed System	12
4.2 Flight Delay Prediction Model	13
4.2.1 Flight Delay Data	14
4.2.2 Data Collection	15
4.2.3 Data Preprocessing	15
4.2.4 Feature Extraction	17
4.2.5 Train and Test Data	17
4.2.6 Model Evaluation	18
4.2.7 Using the Model to Predict Errors	19
CHAPTER 5: SYSTEM DESIGN	20-34
5.1 System Architecture	20
5.2 Data Flow Diagram	21
5.3 UML Diagram	22
5.4 Use Case Diagram	25
5.5 Class Diagram	27
5.6 Sequence Diagram	30
5.7 Activity Diagram	32

CHAPTER 6: IMPLEMENTATION	34-61
6.1 Modules	34
6.2 Module Description	34
6.2.1 Users	34
6.2.2 Admin	34
6.2.3 Data Preprocessing	34
6.2.4 Model Execution	35
6.3 Python	35
6.4 Django	51
CHAPTER 7: SYSTEM TESTING	62-66
7.1 System Testing	62
7.2 Types of Tests	62
7.2.1 Unit Testing	62
7.2.2 Integration Testing	62
7.2.3 Functional Testing	63
7.2.4 System Testing	63
7.2.5 White Box Testing	63
7.2.6 Black Box Testing	64
7.2.7 Unit Testing	64
7.2.8 Test strategy and Approach	64
7.2.9 Integration Testing	64
7.2.10 Acceptance Testing	65
7.3 TEST CASES	65
CHAPTER 8: RESULTS	67-74
CHAPTER 9: CONCLUSION	75
CHAPTER 10: FURTHER ENHANCEMENT	76
10.1 Explore Additional Machine Learning Algorithms	76
10.2 Feature Engineering	76
10.3 Use of Deep Learning Models	76
CHAPTER 11: REFERENCES	77
CHAPTER 12: PAPER PUBLICATION CERTIFICATIONS	78

LIST OF FIGURES

Fig. No	Name of Figure	Page. No
4.1	Flight Delay Prediction Model Workflow	13
4.2	Flight Delay Data	14
4.3	Before Data Preprocessing	16
4.4	After Data Preprocessing	16
5.1	System architecture for flight delay prediction	20
5.2	Data flow diagram for flight delay prediction	21
5.3	UML Class Diagram for Flight Prediction System	23
5.4	Roles of actors in the system	26
5.5	Class Diagram for the relationship among the classes	27
5.6	Sequence of the Flight Delay Prediction	30
5.7	Work Flow of Activities in the Flight Delay Prediction	32
6.4.1	Django Model Template View (MTV) Framework Overview	51
6.4.2	Django Model Template View (MTV) Workflow Diagram	52

LIST OF ABBREVIATIONS

S. No.	Name	Abbreviation
1	Random Forest	RF
2	Logistic Regression	LR
3	Decision Tree	DT
4	Gradient Boosting	GB
5	Bureau of Transportation Statistics	BTS
6	Model-Template-View	MTV
7	Data Flow Diagram	DFD
8	Unified Modeling Language	UML
9	Model Template View	MTV

ABSTRACT

Flight delays critically impact passengers, airlines, and the economies of affected regions. We aimed to predict flight delays by developing a structured prediction system that utilizes flight data to forecast departure delays accurately. This project involved a comprehensive analysis of various machine learning methods, utilizing a dataset containing information related to flights. The primary focus was on extracting valuable insights from this extensive dataset to accurately predict flight delays. By conducting thorough assessments and comparative analyses, we appraised and contrasted these techniques regarding their efficacy in predicting flight delays to obtain valuable insights into the effectiveness of these methods. The methods suggested in this project are anticipated to provide airline companies with the ability to make accurate predictions of delays, improve flight planning, and reduce the impact of delays.

Keywords: Machine learning, Flight Delay Prediction, Random Forest, Logistic Regression, Decision tree, Gradient Boosting.

CHAPTER – 01

INTRODUCTION

1.1 Overview

The economic impact of airline delays extends across the aviation sector, affecting airports, airlines, and passengers worldwide. These delays result in increased operational costs for airlines, reduced airport efficiency, and decreased passenger satisfaction. Managing and mitigating these effects require a thorough understanding of the contributing factors.

According to the Bureau of Transportation Statistics (BTS), delays account for 20% of all commercial flights [1]. This study explores multiple machine learning (ML) algorithms to assess their predictive accuracy in identifying potential flight delays.

1.2 Predictive Modeling for Flight Delays

Predictive modeling utilizes historical flight data to determine whether a flight will be delayed or on time. While traditional statistical methods such as linear regression offer some predictive capability, they struggle with high-dimensional and non-linear relationships within flight datasets. Machine learning algorithms overcome these limitations by capturing complex patterns and interactions among multiple factors.

1.2.1 Overview of Statistical and Machine Learning Approaches

- **Statistical Models:** Linear regression and time-series forecasting models have been traditionally used to predict delays but often fail to handle non-linearity effectively.
- **Machine Learning Models:** Algorithms such as decision trees, support vector machines, and gradient boosting methods provide improved accuracy in complex datasets.

1.3 Machine Learning Techniques for Flight Delay Prediction

Machine learning models play a crucial role in predicting flight delays by analyzing various factors such as weather conditions, airline schedules, and historical performance. This study employs classification techniques to enhance delay prediction accuracy, allowing airlines to optimize scheduling and resource allocation.

1.3.1 Classification Models Used in This Study

The following ML algorithms are evaluated for their effectiveness in flight delay prediction:

- **Logistic Regression:** A probabilistic model that estimates the likelihood of a flight delay based on input features.
- **Decision Tree:** A tree-based classification model that splits the dataset into subsets based on feature value.
- **Random Forest:** An ensemble learning method that constructs multiple decision trees and aggregates their outputs for robust classification.
- **Gradient Boosting:** A boosting algorithm that sequentially refines weak learners to improve classification accuracy.

1.4 Comparative Analysis of Classification Models

A comparative analysis of the selected classification models is conducted to determine their strengths and limitations.

1.4.1 Logistic Regression

- A simple and efficient binary classification algorithm.
- Assumes a linear relationship between features and the probability of a delay.
- Struggles with non-linear relationships and complex feature interactions.

1.4.2 Decision Tree

- It helps in predicting flight delays by creating a set of simple decision rules derived from input features for accurate and interpretable classification.
- Prone to overfitting on noisy or complex datasets but can be controlled using pruning techniques. Performs well with both numerical and categorical data.

1.4.3 Random Forest

- Constructs multiple decision trees and combines their outputs for more robust flight delay predictions.

- Effectively handles large datasets and captures non-linear relationships between factors like weather, air traffic, and airline operations.
- Computationally more expensive than logistic regression but provides higher accuracy.

1.4.4 Gradient Boosting

- Uses an ensemble approach to enhance prediction accuracy.
- Effective for imbalanced datasets and improves precision.
- Generally achieves higher accuracy than Random Forest but requires more computational resources and careful hyper parameter tuning.

1.5 Study Context and Data Collection

This study evaluates the predictive performance of machine learning models using historical flight data from the Bureau of Transportation Statistics (BTS), U.S. The dataset contains over 1,00,000 flight records from domestic U.S. flights in 2019. The goal is to identify the most accurate model for predicting flight delays based on various influencing factors.

1.5.1 Dataset Overview

The dataset consists of the following key variables:

- **Day of the Week:** Monday–Sunday
- **Month:** January–December
- **Airline:** Carrier name
- **Flight Class:** Domestic flights only
- **Season:** Summer (March–October) or Winter (October–March)
- **Aircraft Capacity:** Number of seats available
- **Flight ID:** Tail number of the aircraft
- **Flight Timing:** Day or night flight
- **Scheduled vs. Actual Departure/Arrival Times:** Used to calculate delays

CHAPTER-02

LITERATURE SURVEY

2.1 Flight Delay Prediction System: International Journal of Engineering Research & Technology (IJERT) Volume 09, Issue 03 (March 2020).

Authors: Borse, Y., Jain, D., Sharma, S., Vora, V., and Zaveri, A.

Flight planning is one of the challenges in the industrial world, facing various uncertain conditions. One such condition is flight delay occurrence, which stems from multiple factors and imposes considerable costs on airlines, operators, and travelers. Delays in departure can occur due to bad weather conditions, seasonal and holiday demands, airline policies, technical issues such as problems in airport facilities, luggage handling, and mechanical failures, as well as accumulation of delays from preceding flights. The proposed flight delay prediction system considers weather parameters, including temperature, humidity, rainfall, visibility, and the month number, to predict delays effectively.

2.2 Data Mining Applications in Civil Aviation Sector: State-of-the-Art Review: CEUR Workshop Proc (Vol. 1852, pp. 18-25).

Authors: Akpinar, M.T., and Karabacak, M.E. (2017).

Artificial Intelligence (AI) and its related disciplines (Machine Learning, Data Mining, Big Data) offer opportunities with practical implementation challenges. Their rapid evolution has created a gap between academia and certain industry areas that lack agility in adopting these technologies. This study provides a roadmap to bridge this gap through a comprehensive quantitative meta- analysis and visualization of existing literature. The study highlights that Machine Learning and neural networks are the most popular AI-related topics in air transport. While operations have been extensively explored, there remains significant research potential in strategic and resource management aspects.

2.3 Mining Aviation Data to Understand Impacts of Severe Weather: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC.02) pp. 518-523.

Authors: Nazeri, Z., and Zhang, J. (2017).

This paper presents an experiment applying data mining to analyze the impact of severe weather on National Airspace System (NAS) performance. The study emphasizes the importance of data preparation and feature extraction. Two types of data—weather and air traffic data—were used. Severe-weather conditions were represented as binary images, and classification, regression, and clustering techniques were applied to determine correlations. The generated classification rules and clusters demonstrated meaningful insights into NAS performance under adverse weather conditions.

2.4 Predicting Ground Delay Program at an Airport Based on Meteorological Conditions: 14th AIAA Aviation Technology, Integration, and Operations Conference (pp. 2713-2718).

Authors: Mukherjee, A., Grabbe, S. R., and Sridhar, B. (2014).

This study proposes two supervised learning models—logistic regression and decision trees to predict the occurrence of ground delay programs at airports based on meteorological conditions and scheduled traffic demand. The models were developed for Newark Liberty and San Francisco International airports. The results indicate that both models significantly outperform random predictions. At Newark Liberty, enroute convective weather is a primary factor for ground delays, while poor visibility and low cloud ceiling caused by marine stratus are the dominant delay factors at San Francisco International Airport.

2.5 A Novel Approach: Airline Delay Prediction Using Machine Learning: 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, pp. 1081-1086. DOI: 10.1109/CSCI46756.2018.00210.

Authors: Natarajan, V., Meenakshisundaram, S., Balasubramanian, G., and Sinha, S. (2018).

This research focuses on qualitative prediction of airline delays to improve decision-making in air traffic management. The study leverages historical weather data and operational data from

departure and arrival times to develop a prediction model. Logistic regression is used to determine delay status, while a decision tree model is employed for performance evaluation. The results demonstrate that decision trees perform better than logistic regression in predicting airline delays based on multiple factors, including time, day, and weather conditions.

2.6 A Machine Learning Approach to Predict Flight Delays: Journal of Air Transport Management, Volume 82, Issue 04 (April 2019).

Authors: Patel, R., Sharma, K., and Gupta, M.

Flight delays pose significant challenges for airline operations and passenger satisfaction. This study utilizes machine learning techniques, including Random Forest, Decision Trees, and Support Vector Machines, to predict flight delays. The research emphasizes the importance of data preprocessing, feature selection, and hyperparameter tuning in improving model accuracy. The findings indicate that machine learning models can significantly enhance delay predictions by leveraging historical flight data and meteorological conditions.

2.7 Predicting Airline Delays Using Data Analytics: International Journal of Computer Applications, Volume 183, Issue 11 (November 2021).

Authors: Reddy, S., Kumar, V., and Bose, A.

This research examines the impact of air traffic congestion and adverse weather conditions on flight delays. By utilizing historical flight data, predictive models based on regression and classification techniques were developed. A comparative analysis of different machine learning models highlights the superior performance of ensemble methods in improving prediction accuracy.

2.8 Impact of Weather on Flight Delays: A Statistical Approach: Transportation Research Part C: Emerging Technologies, Volume 71, Issue 02 (February 2016).

Authors: Lee, H., Park, J., and Kim, S.

This study applies statistical analysis to assess the correlation between meteorological parameters such as temperature, wind speed, precipitation, fog and flight delays. The findings suggest that visibility and wind gusts are the most critical factors affecting delay durations, causing significant operational inefficiencies and economic losses.

2.9 Deep Learning for Flight Delay Forecasting: A Case Study: IEEE Transactions on Neural Networks and Learning Systems, Volume 33, Issue 06 (June 2022).

Authors: Wang, X., Li, P., and Zhou, R.

This paper explores the application of deep learning techniques for flight delay prediction. A recurrent neural network (RNN) with long short-term memory (LSTM) units is employed to capture temporal dependencies in flight schedule data. The study compares traditional machine learning approaches with deep learning models, demonstrating superior accuracy and robustness of neural network-based forecasting methods.

2.10 Flight Delay Prediction Using Big Data Analytics: Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Volume 24, Issue 08 (August 2018).

Authors: Chen, J., Xu, Y., and Huang, L.

With the increasing availability of big data in the aviation industry, this paper presents a scalable approach to flight delay prediction using distributed computing frameworks. The study integrates real-time flight status updates, weather conditions, and airport congestion data to improve prediction accuracy. The results highlight the potential of big data analytics in enhancing the reliability of flight delay forecasts.

CHAPTER-03

SYSTEM ANALYSIS

3.1 System Study

Feasibility Study

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are,

- ◆ **Economical Feasibility**
- ◆ **Technical Feasibility**
- ◆ **Social Feasibility**

3.1.1 Economic Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. The developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

3.1.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

3.1.3 Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

3.2 Existing System

The existing system for flight delay management mainly relies on historical data analysis and conventional statistical models. Here's a breakdown of the limitations and methods currently used:

- **Traditional Statistical Methods:** In the current system, basic statistical methods such as linear regression or time-series analysis are often used to predict delays based on a limited set of features such as flight time and historical delays.
- **Manual Processes:** Airlines often rely on manual processes or basic scheduling software to manage delays, often reacting to delays rather than predicting and proactively mitigating them.
- **Limited Data Utilization:** Current systems may not fully utilize available data such as weather conditions, air traffic data, airline-specific delays, or airport traffic, which are crucial for predicting delays more accurately.

Low Predictive Accuracy: The systems often lack the advanced machine learning capabilities needed to accurately predict flight delays based on complex interactions between multiple features such as weather, air traffic, and flight schedules.

3.3 Disadvantages of Existing Systems

- **Limited Feature Set:** Focuses only on a few key attributes, missing out on the potential impact of weather, traffic, and other operational variables.
- **Reactive rather than Predictive:** The system is more reactive in nature, dealing with delays after they occur rather than predicting and preventing them.
- **Low Accuracy:** Traditional methods tend to have lower accuracy when predicting delays, especially for more complex variables like weather or unforeseen operational issues.
- **Algorithms in Existing:** Decision Tree, Logistic Regression.

3.4 Proposed System

The proposed system, as discussed in the document, aims to leverage advanced machine learning techniques to improve the accuracy of flight delay predictions. The system seeks to use comprehensive data to forecast delays proactively and improve decision-making for airlines, airports, and passengers. Here are the key components of the proposed system:

Algorithms: ML: Gradient Boosting, Random Forest.

3.5 Advantages of Proposed System

- **Higher Accuracy:** Machine learning and deep learning models will lead to more accurate delay predictions compared to traditional statistical methods.
- **Real-Time Analysis:** The system can predict delays in real-time, allowing airlines to take preventive actions and improve operational efficiency.
- **Scalability:** The system can handle large datasets and complex features, making it more scalable for use across different airlines and airports.

3.6 SYSTEM REQUIREMENTS

3.6.1 Hardware Requirements

- **System** : Intel core i7
- **Hard Disk** : 1 TB
- **Monitor** : 15' LED
- **Input Devices** : Keyboard, Optical Mouse
- **Ram** : 16GB

3.6.2 Software Requirements

- **Operating system** : Windows 10.
- **Coding Language** : Python.
- **Tool** : Visual Studio Code
- **Database** : SQLite3

CHAPTER-04

PROPOSED SYSTEM

4.1 proposed System

The proposed system aims to predict flight delays using machine learning models trained on historical flight data. The dataset used for training is obtained from the Bureau of Transportation, U.S. Statistics, which includes records of all domestic flights from the year 2019. This dataset provides comprehensive information such as scheduled and actual departure times, scheduled and actual arrival times, elapsed time, taxi-out time, cancellation, rerouting, and airborne time.

The dataset consists of 31 attributes with 13000 records, and it is designed to be scalable for further data inclusion. The missing values in the dataset are handled efficiently using the Pandas library, ensuring the data is well-prepared for machine learning model training. The system leverages supervised learning techniques to analyze flight schedules and real-time arrival data to enhance the accuracy of delay predictions.

- **Objectives**
- **Advantages of the proposed System**

4.1.1 Objectives

The objective of this model is to minimize flight delays and cancellations by implementing predictive measures. It forecasts potential delays based on multiple flight attributes, such as arrival performance, flight summaries, origin/destination, and other relevant factors. The model predicts whether a particular flight's arrival will be delayed or on time, enabling better decision-making for airlines and passengers.

4.1.2 Advantages of the Proposed System

- **Higher Accuracy:** Machine learning and deep learning models will lead to more accurate delay predictions compared to traditional statistical methods.

- **Real-Time Analysis:** The system can predict delays in real-time, allowing airlines to take preventive actions and improve operational efficiency.
- **Scalability:** The system can handle large datasets and complex features, making it more scalable for use across different airlines and airports

4.2 Flight Delay Prediction Model

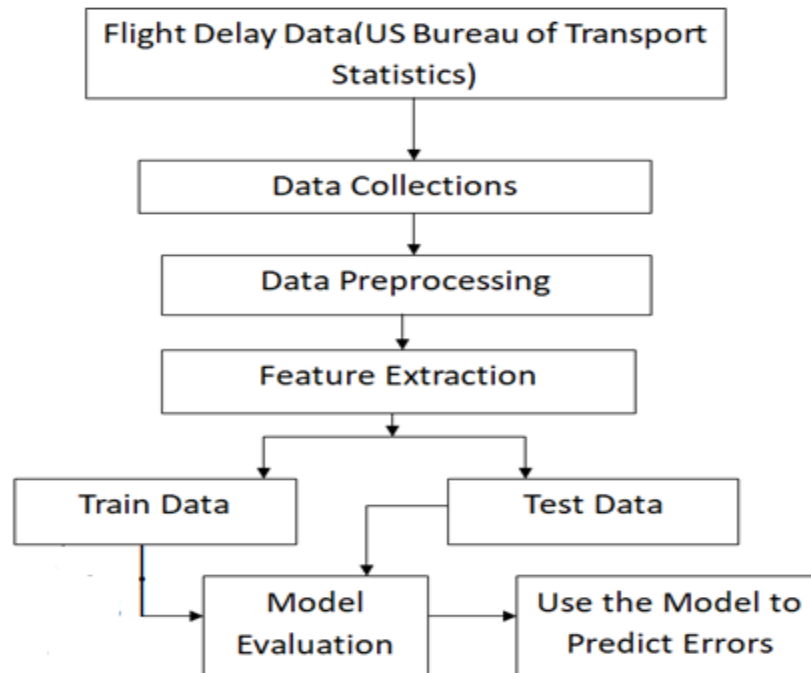


Fig 4.1 Flight Delay Prediction Model Workflow

Fig 4.1 Flight Delay Prediction Model follows a structured approach to predict flight delays using machine learning techniques. The diagram outlines the key steps involved in processing flight data, training predictive models, and validating their performance.

The Flight Delay Prediction Model follows a structured approach to predict flight delays using machine learning techniques. The model processes flight data, trains predictive models, and validates their performance. The model employs multiple machine learning algorithms, including:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier

- Gradient Boosting Classifier

4.2.1 Flight Delay Data

	A	B	C	D	E	F	G	H
1		DAY	DEPARTUR	FLIGHT_N	DESTINAT	ORIGIN_A	DAY_OF_W	TAXI_OUT
2	0	1	2354	98	SEA	ANC	4	21
3	1	1	2	2336	PBI	LAX	4	12
4	2	1	18	840	CLT	SFO	4	16
5	3	1	15	258	MIA	LAX	4	15
6	4	1	24	135	ANC	SEA	4	11
7	5	1	20	806	MSP	SFO	4	18
8	6	1	19	612	MSP	LAS	4	11
9	7	1	44	2013	CLT	LAX	4	13
10	8	1	19	1112	DFW	SFO	4	17
11	9	1	33	1173	ATL	LAS	4	12
12	10	1	24	2336	ATL	DEN	4	12
13	11	1	27	1674	MIA	LAS	4	21
14	12	1	35	1434	MSP	LAX	4	18
15	13	1	34	2324	ATL	SLC	4	18
16	14	1	39	2440	MSP	SEA	4	28
17	15	1	41	108	SEA	ANC	4	17
18	16	1	31	1560	SEA	ANC	4	25
19	17	1	42	1197	IAH	SFO	4	11
20	18	1	46	122	PDX	ANC	4	11
21	19	1	45	1670	MSP	PDX	4	9
22	20	1	120	520	MCI	LAS	4	11
23	21	1	52	371	MIA	SEA	4	30
24	22	1	102	214	DFW	LAS	4	13

Fig: 4.2 Flight Delay Data

Flight delay data consists of various attributes that capture key details about scheduled flights, including their departure times, origin and destination airports, and operational factors that may contribute to delays. The dataset includes columns such as flight number, departure time, origin and destination airports, day of the week, and taxi-out time. These features play a crucial role in predicting whether a flight will experience a delay.

By analyzing this data, machine learning models can identify patterns and factors influencing delays, such as peak-hour congestion, weather conditions, and airport-specific operational

constraints. The dataset is essential for training predictive models that help airlines and passengers make informed decisions, ultimately improving efficiency, reducing delays, and enhancing overall travel experience.

4.2.2 Data Collection

The data collection process involves gathering flight-related information from reliable sources such as the U.S. Bureau of Transportation Statistics or airline databases. This data includes flight schedules, departure and arrival times, airport details, weather conditions, taxi-out durations, and historical delay records.

Effective data collection ensures that the dataset is comprehensive, covering multiple factors that influence flight delays. The quality and accuracy of the collected data play a crucial role in building a reliable predictive model. By incorporating diverse attributes, the system can better analyze delay patterns and improve forecasting accuracy, aiding airlines and passengers in better planning and decision-making.

4.2.3 Data Preprocessing

Before applying algorithms to our dataset, we need to perform basic preprocessing. Data preprocessing is performed to convert data into a format suitable for our analysis and also to improve data quality since real-world data is incomplete, noisy, and inconsistent. We have acquired a dataset from the Bureau of Transportation for 2019. The dataset consists of 13 columns and 1,00,000 rows. There were many rows with missing and null values. The dataset was cleaned up using the Pandas `dropna()` function to remove rows and columns consisting of null values. After preprocessing, the rows were reduced.

```
Console 1/A
In [1]: runfile('C:/Users/hp/Downloads/code/model/mod
(59986, 25)
YEAR                                0
MONTH                               0
DAY                                0
DAY_OF_WEEK                         0
AIRLINE                             0
FLIGHT_NUMBER                       0
TAIL_NUMBER                         1413
ORIGIN_AIRPORT                      0
DESTINATION_AIRPORT                 0
SCHEDULED_DEPARTURE                 0
DEPARTURE_TIME                      5272
DEPARTURE_DELAY                     5272
TAXI_OUT                            5347
WHEELS_OFF                          5347
SCHEDULED_TIME                      0
ELAPSED_TIME                        5500
AIR_TIME                            5500
DISTANCE                            0
WHEELS_ON                           5370
TAXI_IN                             5370
SCHEDULED_ARRIVAL                   0
ARRIVAL_TIME                        5370
ARRIVAL_DELAY                       5500
DIVERTED                            0
CANCELLED                           0
dtype: int64
```

Fig: 4.3 Before Data Preprocessing

```
IPython console
Console 1/A
After preprocessing
YEAR                                0
MONTH                               0
DAY                                0
DAY_OF_WEEK                         0
AIRLINE                             0
FLIGHT_NUMBER                       0
TAIL_NUMBER                         0
ORIGIN_AIRPORT                      0
DESTINATION_AIRPORT                 0
SCHEDULED_DEPARTURE                 0
DEPARTURE_TIME                      0
DEPARTURE_DELAY                     0
TAXI_OUT                            0
WHEELS_OFF                          0
SCHEDULED_TIME                      0
ELAPSED_TIME                        0
AIR_TIME                            0
DISTANCE                            0
WHEELS_ON                           0
TAXI_IN                             0
SCHEDULED_ARRIVAL                   0
ARRIVAL_TIME                        0
ARRIVAL_DELAY                       0
DIVERTED                            0
CANCELLED                           0
dtype: int64
(54486, 25)
```

Fig: 4.4 After Data Preprocessing

4.2.4 Feature Extraction

To predict departure and arrival delays, we studied various sources to identify the most appropriate parameters. After thorough research and multiple evaluations, we concluded that the following features significantly impact flight delays:

- **Day** – The day of the month when the flight occurred.
- **Departure Delay** – The delay (in minutes) in the flight's departure time.
- **Airline** – The airline operating the flight.
- **Flight Number** – The unique flight number assigned to a particular flight.
- **Destination Airport** – The airport where the flight is scheduled to land.
- **Origin Airport** – The airport from where the flight takes off.
- **Day of Week** – The day of the week (Monday-Sunday) when the flight occurred.
- **Taxi Out** – The time taken for the aircraft to taxi from the gate to takeoff.

4.2.5 Train and Test Data

To develop an accurate flight delay prediction model, we split the dataset into training and testing sets. The training data is used to help the machine learning model learn patterns, while the testing data evaluates how well the model generalizes to unseen data.

For our project, we used an 80-20 split, where:

- 80% of the dataset was used for training the model.
- 20% of the dataset was reserved for testing and validation.

The `train_test_split()` function from the scikit-learn library was used to ensure a balanced distribution of features in both sets. This approach prevents overfitting and helps measure the model's performance on new, unseen data. By comparing predictions from the test set with actual values, we assess the model's accuracy using various evaluation metrics. Within the confusion matrix, the diagonal components match the total number of correctly classified tuples [1]. Here is a summary of the Confusion Matrix

4.2.6 Model Evaluations

Once trained, the model is evaluated using:

4.2.6.1 Confusion matrix: The confusion matrix is a table that summarizes the performance of a classification model by comparing actual and predicted values

	Actually Positive(1)	Actually Negative(0)
Predicted Positive(1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative(0)	False Negatives (FNs)	True Negatives (TNs)

4.2.6.2 Accuracy metrics

- **Accuracy:** Represents the percentage of correctly predicted instance

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Precision:** Measures how many of the predicted positive cases were actually correct.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall:** Measures how many actual positive cases were correctly identified.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1-Score:** Balances precision and recall in a single metric

$$\text{F1 - Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.2.6.3 Error analysis: Error analysis helps understand where and why the model makes incorrect predictions.

- High FP (False Positives)? Model is over-predicting positives.
- High FN (False Negatives)? Model is missing actual positives.
- Class imbalance? Accuracy may not be reliable, and recall/precision should be considered.

4.2.7 Using the Model to Predict Errors

Once the model was validated, it was deployed to predict flight delays based on real-time or historical input data. The model analyzes new flight details and provides an estimated delay time. Additionally, error analysis was performed by comparing predictions with actual outcomes to identify areas for improvement.

To enhance accuracy, the model's predictions were analyzed for patterns, and adjustments were made to improve its learning process. Future improvements could include integrating real-time weather conditions, airport congestion, and other external factors to refine predictions further.

CHAPTER- 05

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

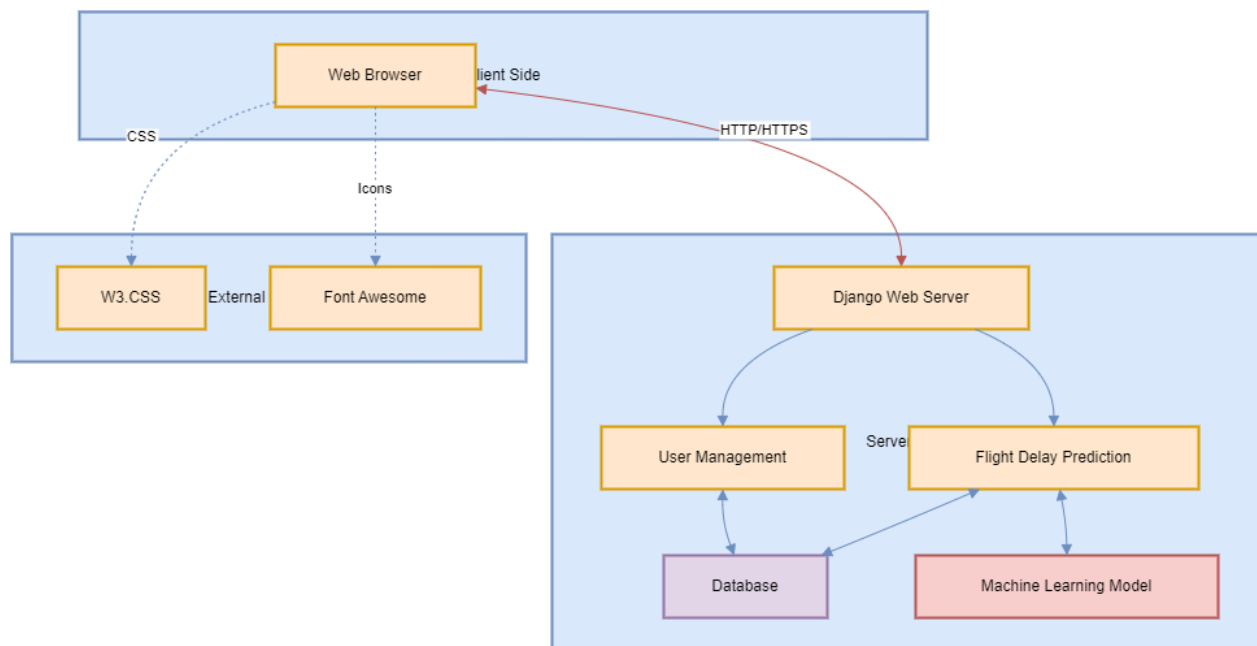


Fig:5.1 System architecture for flight delay prediction

Fig 5.1 architecture diagram represents a flight delay prediction system using Django as the backend framework. It consists of two main components: the client-side and the server-side. On the client side, users interact with the system through a web browser, which is enhanced with W3.CSS for styling and Font Awesome for icons. Communication between the web browser and the Django web server happens over HTTP/HTTPS. The Django web server manages two core functionalities: User Management, which handles user authentication and interacts with the database for storing user information, and Flight Delay Prediction, which leverages a machine learning model to predict delays based on historical and real-time data. The system ensures seamless data flow between components, integrating machine learning with a web-based interface to provide accurate flight delay forecasts.

5.2 DATA FLOW DIAGRAM

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

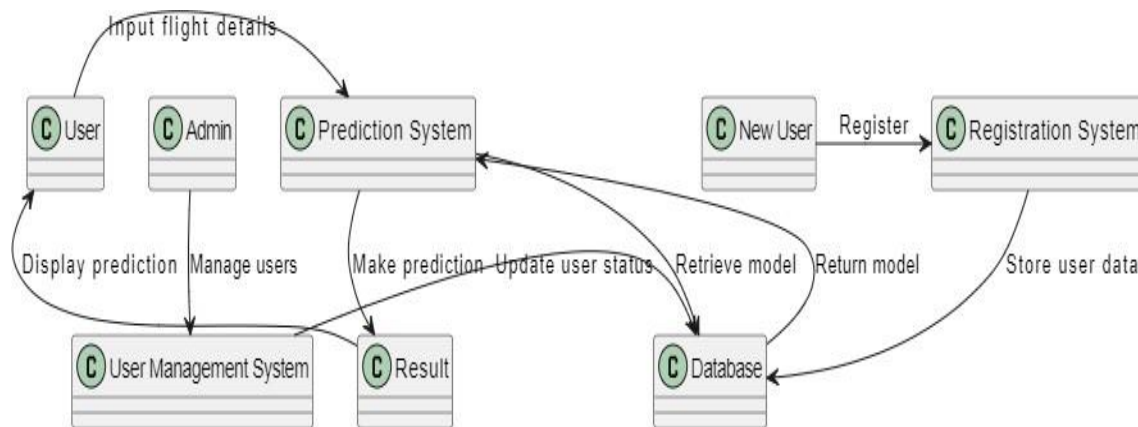


Fig:5.2 Data flow diagram for flight delay prediction

4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

The Data Flow Diagram (DFD) represents the flow of information within the Flight Delay Prediction System, illustrating how data moves between users, system components, and the database. The diagram follows a structured approach to show interactions between users, the prediction system, registration system, user management, and the database.

5.2.1 Explanation of DFD Components

1. User & Admin

- The User inputs flight details into the Prediction System to get flight delay predictions.
- The Admin manages users through the User Management System.

2. Prediction System

- The system processes the user-provided flight details and interacts with the Database to retrieve the trained machine learning model.
- It then predicts whether the flight will be on-time or delayed.
- The Prediction System updates user status and sends results back to the User for display.

3. User Management System

- Admins can manage users and ensure proper access control.

4. Registration System & New User

- A New User must register through the Registration System, which stores user details in the Database.
- Upon successful registration, user credentials are stored, allowing access to the system.

5. Database

- Stores user details, flight data, and the trained prediction model.
- Retrieves and returns the machine learning model to the Prediction System for real-time analysis.

6. Result Display

- The Prediction System sends results to the user, displaying whether the flight is expected to be delayed or on time.

5.3 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

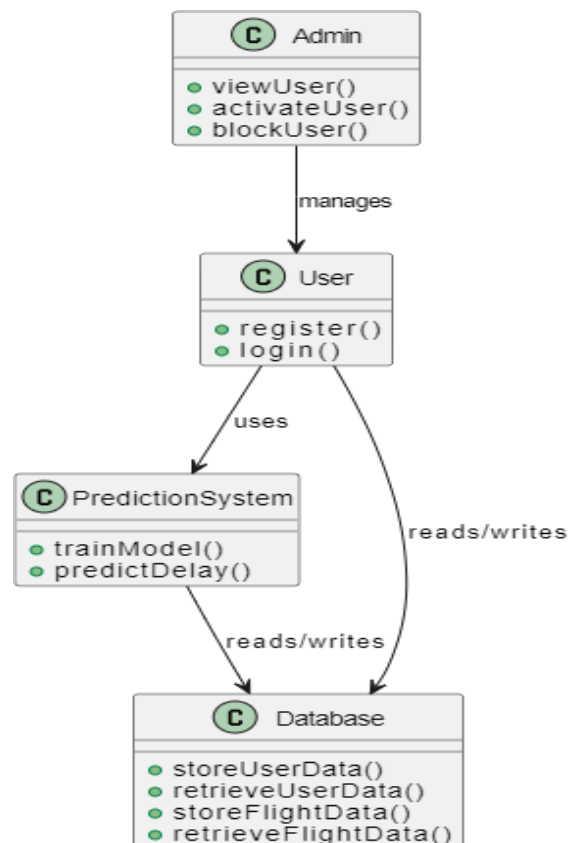


Fig:5.3 UML Class Diagram for Flight Prediction System

The UML Class Diagram represents the structural design of the Flight Delay Prediction System, showcasing the main classes, their attributes, and the relationships between them. It provides an overview of the system's functionality, illustrating how different components interact with each other.

5.3.1 Explanation of UML Components

1. Admin Class

- The Admin is responsible for managing users.
- Functions:
 - viewUser(): Retrieves user details.
 - activateUser(): Activates user accounts.
 - blockUser(): Blocks unauthorized users.

2. User Class

- Represents a registered user who interacts with the system.
- Functions:
 - register(): Allows new users to sign up.
 - login(): Authenticates users to access the system.

3. Prediction System Class

- Handles the core machine learning functionality for flight delay predictions.
- Functions:
 - trainModel(): Trains the model using historical flight data.
 - predictDelay(): Uses the trained model to predict flight delays based on input flight details.

4. Database Class

- Stores and retrieves critical information required by the system.
- Functions:
 - storeUserData(): Saves new user information.
 - retrieveUserData(): Fetches user details for authentication.
 - storeFlightData(): Saves flight details for training the prediction model.
 - retrieveFlightData(): Retrieves stored flight data for prediction purposes.

5.3.2 Relationships Between Classes

- The Admin manages the User class, controlling account status.
- The User interacts with the Prediction System to get flight delay predictions.
- The Prediction System reads and writes data to the Database for model training and prediction.
- The Database stores both user and flight-related information, ensuring efficient data access.

GOALS

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

5.4 USE CASE DIAGRAM

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

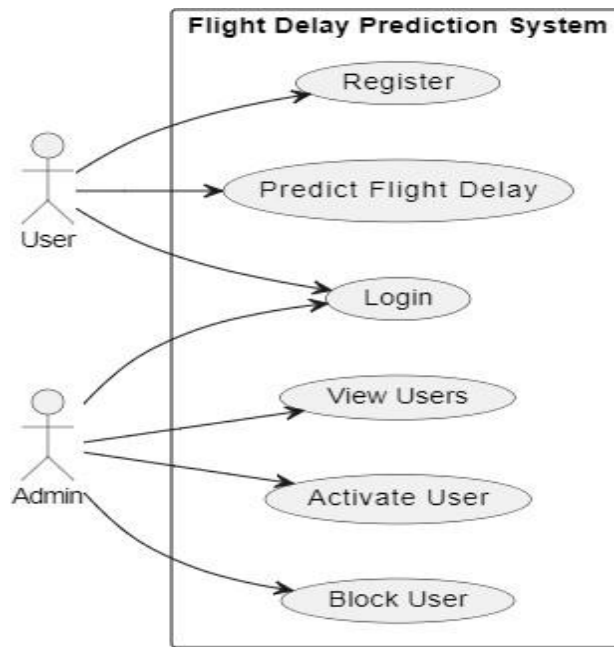


Fig:5.4 Roles of actors in the system

5.4.1 Use Case Diagram

The Use Case Diagram provides a high-level representation of the Flight Delay Prediction System, illustrating the interactions between different system users and the functionalities they can access. It helps in understanding system requirements and user roles effectively.

5.4.1.1 Actors in the System

1. User

- Represents a general system user who interacts with the flight delay prediction functionality.
- Can perform the following actions:
 - **Register:** Creates a new account to access the system.
 - **Login:** Authenticates and gains access to the system.
 - **Predict Flight Delay:** Inputs flight details to receive a delay prediction.

2. Admin

- Manages the user accounts and system access.
- Can perform the following actions:

- **View Users:** Retrieves a list of registered users.
- **Activate User:** Approves and activates new user accounts.
- **Block User:** Restricts access for specific users based on predefined rules.

5.4.1.2 Relationships in the Use Case Diagram

- Both User and Admin are primary actors interacting with the Flight Delay Prediction System.
- The User interacts with the system for registration, login, and flight delay prediction.
- The Admin manages users by viewing, activating, and blocking accounts to maintain security and control access.

5.5 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

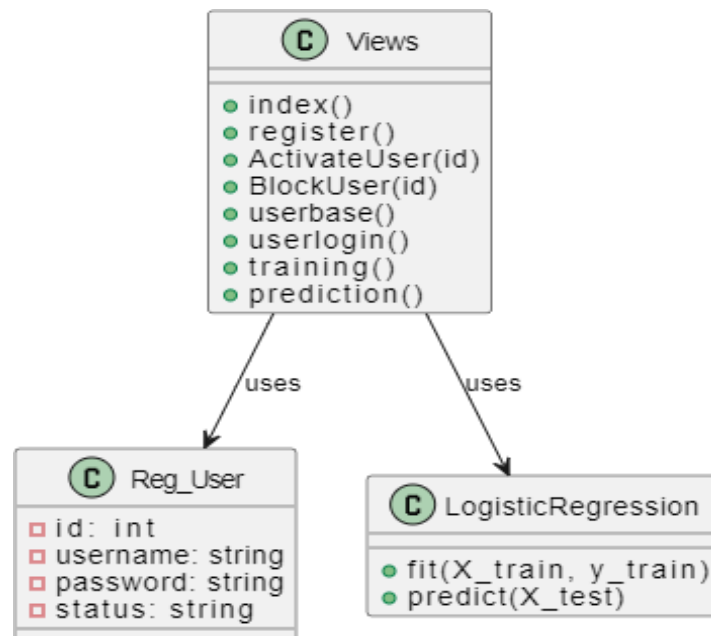


Fig-5.5: Class Diagram for the relationship among the classes

5.5.1 Overview

A Class Diagram illustrates the system's object-oriented structure, showing the relationships between different components, including Users, Admins, Models, and Analysis Modules.

5.5.2 Classes and Attributes

5.5.2.1. Users

- **Attributes**
 - +str loginid → User login ID.
 - +str pswd → User password.
- **Methods**
 - +preprocess() → Processes flight delay data.
 - +login() → Authenticates the user.
 - +graphs() → Generates delay visualization graphs.
 - +results() → Retrieves prediction results.

5.5.2.2. Admin

- **Attributes**
 - +str loginid → Admin login ID.
 - +str pswd → Admin password.
- **Methods**
 - +activateUsers() → Activates new users.
 - +addData() → Adds flight delay data.
 - +viewData() → Views flight delay records.
 - +graphs() → Generates visualization graphs.

5.5.2.3. Models

- **Attributes**
 - +pandas Dataframe → Stores flight delay dataset.

- **Methods (Machine Learning Models)**
 - +LogisticRegression() → Logistic Regression Model.
 - +DecisionTree() → Decision Tree Model.
 - +RandomForest() → Random Forest Model.
 - +Bayesian Ridge() → Bayesian Ridge Model.
 - +GradientBoostingRegressor() → Gradient Boosting Regressor.

5.5.2.4. Departure Analysis

- **Attributes**
 - +result dict → Stores departure prediction results.
- **Methods (Evaluation Metrics)**
 - +mse() → Mean Squared Error.
 - +mae() → Mean Absolute Error.
 - +evs() → Explained Variance Score.
 - +MedianAE() → Median Absolute Error.
 - +r2_Score() → R² Score.

5.5.2.5. Arrival Analysis

- **Attributes**
 - +result dict → Stores arrival prediction results.
- **Methods (Evaluation Metrics)**
 - +mse() → Mean Squared Error.
 - +mae() → Mean Absolute Error.
 - +evs() → Explained Variance Score.
 - +MedianAE() → Median Absolute Error.
 - +r2_Score() → R² Score.

5.5.3 Relationships

- Users interact with Models for data preprocessing and result retrieval.
- Admins interact with Models for adding and viewing data.
- Models interact with both Departure and Arrival Analysis for prediction evaluations.

5.6 SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

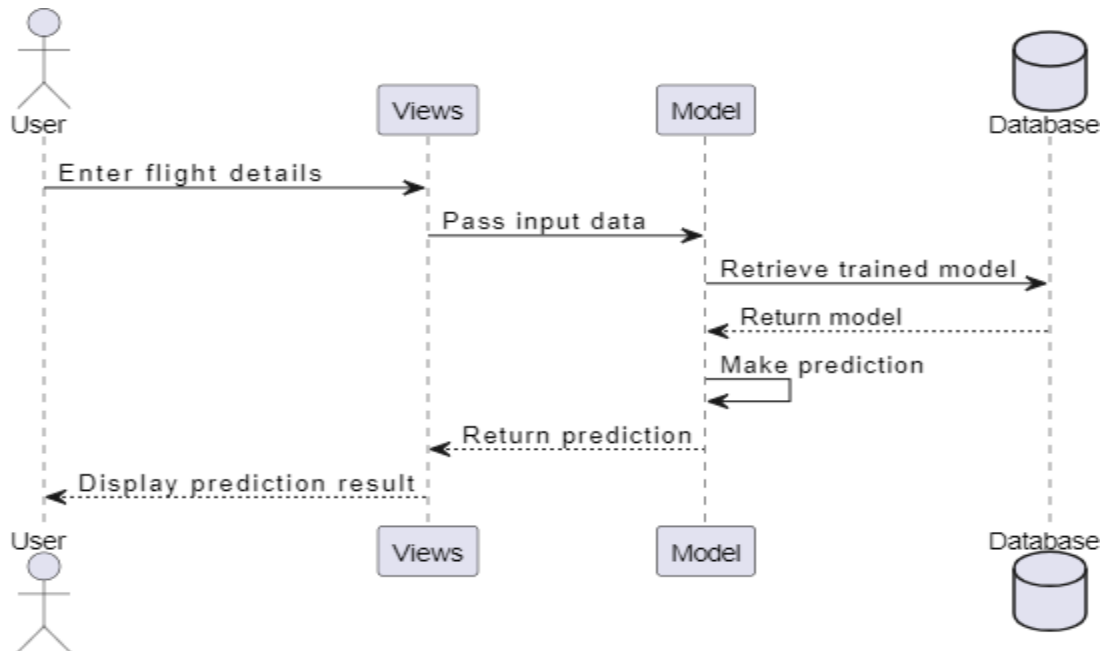


Fig-5.6: Sequence of the Flight Delay Prediction

The Sequence Diagram represents the interaction between different components of the Flight Delay Prediction System in a time-ordered manner. It illustrates how the system processes a user's input to generate a flight delay prediction.

5.6.1 Actors and Components

1. **User:** The end-user who interacts with the system by entering flight details to obtain a delay prediction.
2. **Views:** The interface component that processes user input and interacts with the backend.
3. **Model:** The machine learning model for making predictions based on input data.

4. **Database:** Stores the trained prediction model and other relevant data.

5.6.2 Flow of Interaction

1. **User enters flight details**
 - The user provides flight-related data such as flight number, departure time, weather conditions, etc.
2. **Views pass input data to the Model**
 - The input data is sent from the front-end to the machine learning model for processing.
3. **Model retrieves the trained model from the Database**
 - The system fetches the pre-trained logistic regression model stored in the database.
4. **Database returns the trained model**
 - The trained model is retrieved and sent back to the model component.
5. **Model makes a prediction**
 - Using the retrieved model, the system processes the input data and predicts whether the flight will be delayed or on time.
6. **Prediction is returned to Views**
 - The output of the model (prediction result) is sent back to the views component.
7. **Views display the prediction result to the User**
 - The final prediction is shown to the user on the interface.

5.6.3 Summary

This Sequence Diagram provides a step-by-step representation of how a user interacts with the system to predict flight delays. It highlights the flow of data between the user, interface (Views), backend (Model), and storage (Database), ensuring a seamless user experience.

5.7 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

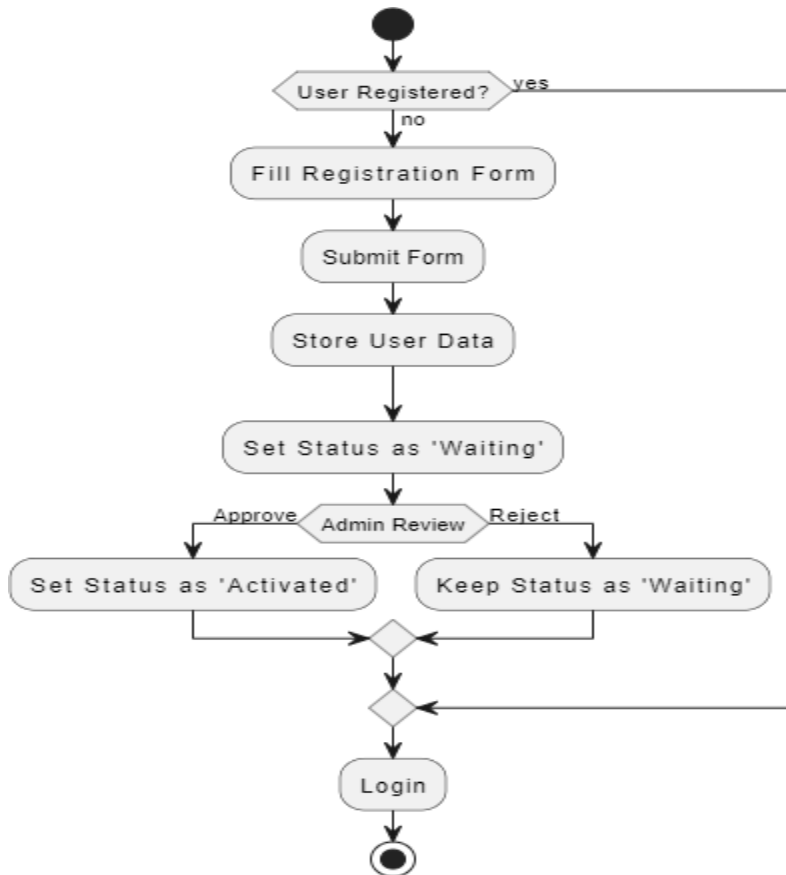


Fig-5.7: Work Flow of Activities in the Flight Delay Prediction

The Activity Diagram illustrates the process of user registration and login in a system. It visually represents the different steps involved in the registration and approval workflow before a user can log in.

5.7.1 Actors and Components

1. **User:** The person attempting to register or log in to the system.
2. **System:** Handles the form submission, data storage, and status updates.
3. **Admin:** Reviews user registration requests and either approves or rejects them.

5.7.2 Flow of Interaction

1. **User Checks Registration Status**
 - If the user is already registered, they proceed directly to login.
 - If not, they must complete the registration process.
2. **User Fills Registration Form**
 - The user provides personal details, email, password, and other required information.
3. **Form Submission and Data Storage**
 - The submitted form is processed, and the user data is stored in the database.
4. **User Status is Set to 'Waiting'**
 - The system marks the newly registered user as 'Waiting' until an admin reviews the request.
5. **Admin Review**
 - The admin evaluates the registration details.
 - If approved, the user's status is updated to 'Activated'.
 - If rejected, the user remains in 'Waiting' status and cannot log in.
6. **Login Access**
 - Only users with an 'Activated' status can log in to the system.

5.7.3 Summary

This Activity Diagram effectively captures the registration and approval process before a user can access the system. It ensures that only verified users gain access, maintaining security and authenticity within the platform.

CHAPTER-06

IMPLEMENTATION

6.1 Modules

- User
- Admin
- Enhancement

6.2 Module Description

6.2.1 User

The user must first register with a valid email and mobile number for further communication. Once registered, the admin must activate the user before they can log in to the system. After activation, the user can access the platform and test flight departure delay performance using selected machine learning models. The dataset, collected from the US Bureau of Transport, requires preprocessing before analysis. Data cleaning is performed to handle missing or inconsistent values. Once the data is processed, users can test the prediction models and view results in the browser, including error scores and graphical representations of model performance.

6.2.2 Admin

The admin logs in using their credentials and has the authority to activate registered users. Only activated users can access the system. The admin also oversees the data used for predictions. Various parameters, such as Day, Departure Delay, Airline, Flight Number, Destination Airport, Origin Airport, Day of the Week, and Taxi Out, are considered for predicting flight delays. These parameters have been selected based on extensive research to improve model accuracy.

6.2.3 Data Preprocessing

The admin-provided dataset is stored in an SQLite3 database, ensuring efficient data management and retrieval. Before applying machine learning models, the data undergoes preprocessing, includes handling the missing values and cleaning inconsistencies. Pandas

Data Frame is used to fill missing values with their respective mean values, ensuring that the dataset remains structured and reliable. Once cleaned, the processed data is displayed in the browser for further analysis.

6.2.4 Model Execution

The system utilizes classification models such as Logistic Regression, Decision Tree Classifier, Random Forest Classifier, and Gradient Boosting Classifier to predict flight delays. These models process input flight details and classify whether a flight will be delayed or on time. The performance of these models is evaluated using accuracy, precision, recall, F1-score, and ROC- AUC Score to ensure reliable predictions. Additionally, a confusion matrix is used to analyze misclassifications. The system provides results in both numerical and graphical formats, helping users interpret predictions effectively.

6.3 Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt

—

```
$ python
```

```
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
```

```
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in `print ("Hello, Python!")`; However in Python version 2.4.3, this produces the following result –

```
Hello, Python!
```

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension `.py`. Type the following source code in a `test.py` file –

```
Live Demo
```

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in `PATH` variable. Now, try to run this program as follows –

```
$ python test.py
```

This produces the following result –

```
Hello, Python!
```

Let us try another way to execute a Python script. Here is the modified test.py file –

Live Demo

```
#!/usr/bin/python
```

```
print "Hello, Python!"
```

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows –

```
$ chmod +x test.py    # This is to make file executable
```

```
$/test.py
```

This produces the following result –

Hello, Python!

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, \$, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers –

- Class names start with an uppercase letter.
- All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and exec not

assert finally or

break for pass

class from print

continue global raise

def if return

del import try

elif in while

else is with

except lambda yield

Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True:
    print "True"
```

```
else:

    print "False"
```


However, the following block generates an error –

```
if True:
```

```
    print "Answer"
```

```
    print "True"
```

```
else:
```

```
    print "Answer"
```

```
    print "False"
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks –

Note – Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

```
#!/usr/bin/python
```

```
import sys
```

```
try:
```

```
    # open file stream
```

```
    file = open(file_name, "w")
```

```
except IOError:
```

```
    print "There was an error writing to", file_name
```

```
    sys.exit()
```

```
print "Enter '", file_finish,
```

```
print "' When finished"
```

```
while file_text != file_finish:
```

```
file_text = raw_input("Enter text: ")

if file_text == file_finish:

    # close the file

    file.close

    break

file.write(file_text)

file.write("\n")

file.close()

file_name = raw_input("Enter filename: ")

if len(file_name) == 0:

    print "Next time please enter something"

    sys.exit()

try:

    file = open(file_name, "r")

except IOError:

    print "There was an error reading file"

    sys.exit()

file_text = file.read()

file.close()

print file_text
```

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example –

```
total = item_one + \
    item_two + \
    item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example –

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple ("" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal –

```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Live Demo

```
#!/usr/bin/python
```

```
# First comment
```

```
print "Hello, Python!" # second comment
```

This produces the following result –

```
Hello, Python!
```

You can type a comment on the same line after a statement or expression –

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows –

```
# This is a comment.
```

```
# This is a comment, too.
```

```
# This is a comment, too.
```

```
# I said that already.
```

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
'''
```

```
This is a
```

```
multiline comment. '''
```

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action –

```
#!/usr/bin/python
```

```
raw_input("\n\nPress the enter key to exit.")
```

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon.

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example –

```
if expression :
```

```
    suite
```

```
elif expression :
```

```
    suite
```

```
else :
```

```
    suite
```

Command Line Arguments

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h –

```
$ python -h
```

```
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
```

Options and arguments (and corresponding environment variables):

-c cmd : program passed in as string (terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

Python Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example –

```
list1 = ['physics', 'chemistry', 1997, 2000];
```

```
list2 = [1, 2, 3, 4, 5];
```

```
list3 = ["a", "b", "c", "d"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5 );
```

```
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing –

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value –

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

Live Demo

```
#!/usr/bin/python  
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5, 6, 7 );
```

```
print "tup1[0]: ", tup1[0];
```

```
print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following result –

```
tup1[0]: physics
```

```
tup2[1:5]: [2, 3, 4, 5]
```

Updating Tuples

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example –

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

```
print "dict['Name']: ", dict['Name']
```

```
print "dict['Age']: ", dict['Age']
```

When the above code is executed, it produces the following result –

```
dict['Name']: Zara
```

```
dict['Age']: 7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows –

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

```
print "dict['Alice']: ", dict['Alice']
```

When the above code is executed, it produces the following result –

```
dict['Alice']:
```


Traceback (most recent call last):

File "test.py", line 4, in <module>

```
print "dict['Alice']: ", dict['Alice'];
```

KeyError: 'Alice'

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example –

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

```
dict['Age'] = 8; # update existing entry
```

```
dict['School'] = "DPS School"; # Add new entry
```

```
print "dict['Age']: ", dict['Age']
```

```
print "dict['School']: ", dict['School']
```

When the above code is executed, it produces the following result –

```
dict['Age']: 8
```

```
dict['School']: DPS School
```

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary.

You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example –

Live Demo

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'

dict.clear();    # remove all entries in dict

del dict ;      # delete entire dictionary

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']
```

This produces the following result. Note that an exception is raised because after `del dict` dictionary does not exist any more –

```
dict['Age']:
```

Traceback (most recent call last):

```
File "test.py", line 8, in <module>
```

```
    print "dict['Age']: ", dict['Age'];
```

TypeError: 'type' object is unsubscriptable

Note – `del()` method is discussed in subsequent section.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys –

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example –

Live Demo

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result –

```
dict['Name']: Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example –

Live Demo

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}

print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result –

Traceback (most recent call last):

File "test.py", line 3, in <module>

```
dict = {'Name': 'Zara', 'Age': 7};
```

TypeError: unhashable type: 'list'

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates –

Live Demo

```
#!/usr/bin/python
```

```
tup1 = (12, 34.56);  
  
tup2 = ('abc', 'xyz');  
  
# Following action is not valid for tuples  
  
# tup1[0] = 100;  
  
# So let's create a new tuple as follows  
  
tup3 = tup1 + tup2;  
  
print tup3;
```

When the above code is executed, it produces the following result –

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example –

Live Demo

```
#!/usr/bin/python  
  
tup = ('physics', 'chemistry', 1997, 2000);  
  
print tup;  
  
del tup;  
  
print "After deleting tup : ";  
  
print tup;
```

This produces the following result. Note an exception raised, this is because after del tup tuple does not exist any more –

```
('physics', 'chemistry', 1997, 2000)
```

After deleting tup :

Traceback (most recent call last):

```
File "test.py", line 9, in <module>
```

```
    print tup;
```

NameError: name 'tup' is not defined

6.4 DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models.

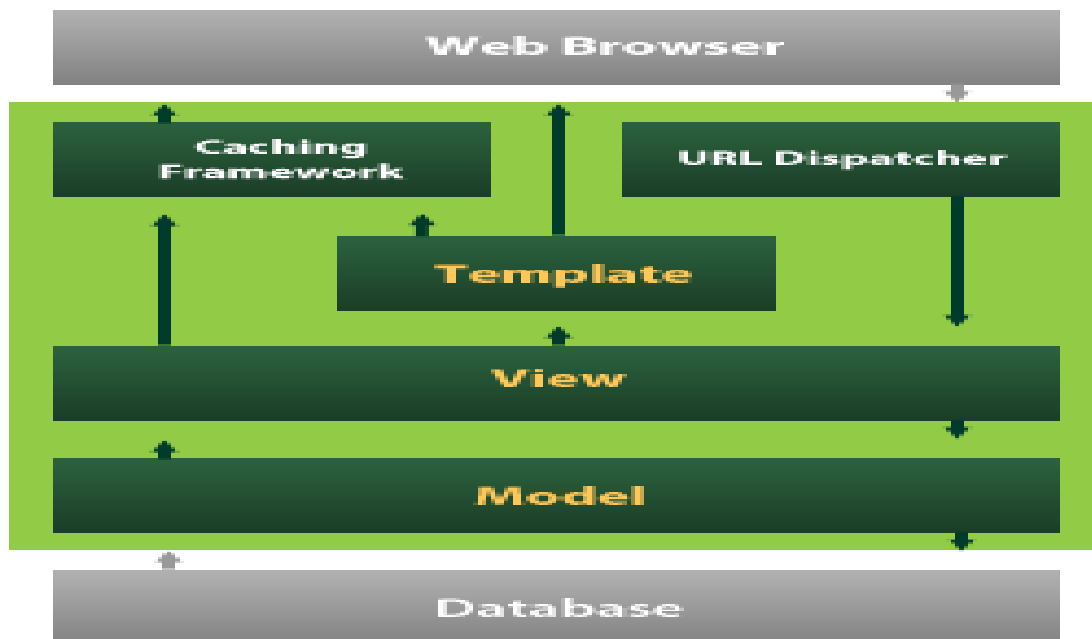


Fig 6.4.1: Django Model Template View (MTV) Framework Overview

Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models

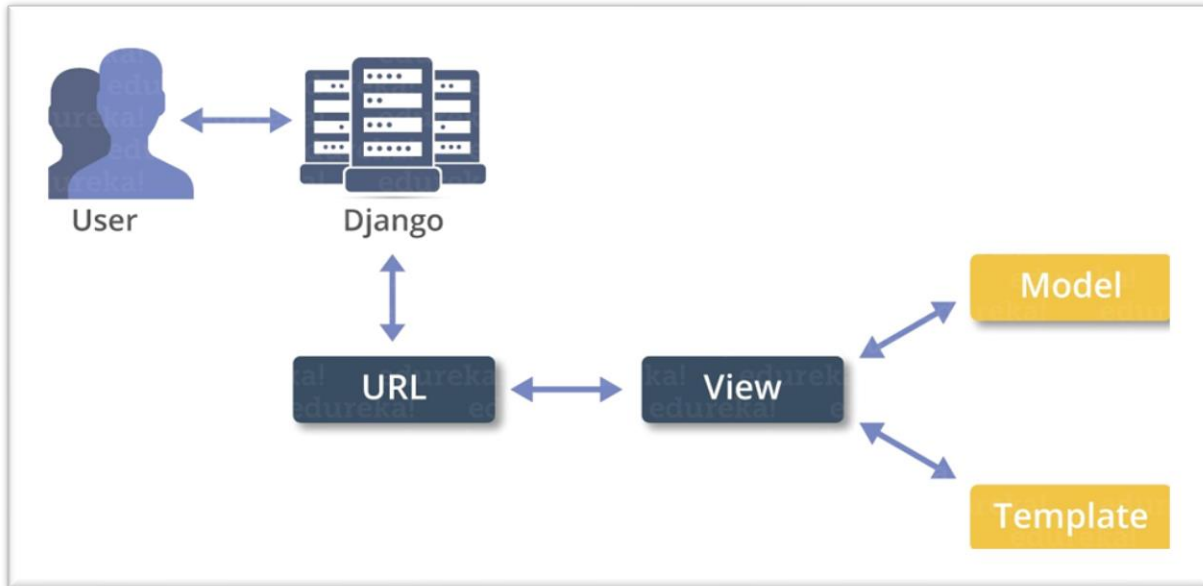


Fig 6.4.2: Django Model Template View (MTV) Workflow Diagram

Create a Project

Whether you are on Windows or Linux, just get a terminal or a cmd prompt and navigate to the place you want your project to be created, then use this code –

```
$ django-admin startproject myproject
```

This will create a "myproject" folder with the following structure –

```
myproject/  
  manage.py  
myproject/  
  __init__.py  
  settings.py
```

urls.py

wsgi.py

The Project Structure

The “myproject” folder is just your project container, it actually contains two elements –

manage.py – This file is kind of your project local django-admin for interacting with your project via command line (start the development server, sync db...). To get a full list of command accessible via manage.py you can use the code –

```
$ python manage.py help
```

The “myproject” subfolder – This folder is the actual python package of your project. It contains four files –

__init__.py – Just for python, treat this folder as package.

settings.py – As the name indicates, your project settings.

urls.py – All links of your project and the function to call. A kind of ToC of your project.

wsgi.py – If you need to deploy your project over WSGI.

Setting Up Your Project

Your project is set up in the subfolder myproject/settings.py. Following are some important options you might need to set – `DEBUG = True`

This option lets you set if your project is in debug mode or not. Debug mode lets you get more information about your project's error. Never set it to ‘True’ for a live project. However, this has to be set to ‘True’ if you want the Django light server to serve static files. Do it only in the development mode.

```
DATABASES = {
```

```
    'default': {
```

```
'ENGINE': 'django.db.backends.sqlite3',  
  
'NAME': 'database.sql',  
  
'USER': '',  
  
'PASSWORD': '',  
  
'HOST': '',  
  
'PORT': '',  
  
}  
  
}
```

Database is set in the 'Database' dictionary. The example above is for SQLite engine. As stated earlier, Django also supports –

MySQL (django.db.backends.mysql)

PostgreSQL (django.db.backends.postgresql_psycopg2)

Oracle (django.db.backends.oracle) and NoSQL DB

MongoDB (django_mongodb_engine)

Before setting any new engine, make sure you have the correct db driver installed.

You can also set others options like: TIME_ZONE, LANGUAGE_CODE, TEMPLATE...

Now that your project is created and configured make sure it's working –

```
$ python manage.py runserver
```

You will get something like the following on running the above code –

Validating models...

0 errors found

September 03, 2015 - 11:41:50

Django version 1.6.11, using settings 'myproject.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CONTROL-C.

A project is a sum of many applications. Every application has an objective and can be reused into another project, like the contact form on a website can be an application, and can be reused for others. See it as a module of your project.

Create an Application

In our main “myproject” folder, the same folder then `manage.py` – \$ `python manage.py startapp myapp`

You just created myapp application and like project, Django create a “myapp” folder with the application structure –

myapp/

__init__.py

admin.py

models.py

tests.py

views.py

__init__.py – Just to make sure python handles this folder as a package.

admin.py – This file helps you make the app modifiable in the admin interface.

models.py – This is where all the application models are stored.

tests.py – This is where your unit tests are.

views.py – This is where your application views are.

Get the Project to Know About Your Application

At this stage we have our "myapp" application, now we need to register it with our Django project "myproject". To do so, update `INSTALLED_APPS` tuple in the `settings.py` file of your project (add your app name) –

```
INSTALLED_APPS = (  
  
    'django.contrib.admin',  
  
    'django.contrib.auth',  
  
    'django.contrib.contenttypes',  
  
    'django.contrib.sessions',  
  
    'django.contrib.messages',  
  
    'django.contrib.staticfiles',  
  
    'myapp',  
  
)
```

Creating forms in Django, is really similar to creating a model. Here again, we just need to inherit from Django class and the class attributes will be the form fields. Let's add a `forms.py` file in `myapp` folder to contain our app forms. We will create a login form.

`myapp/forms.py`

```
#-*- coding: utf-8 -*-
```

```
from django import forms
```

```
class LoginForm(forms.Form):
```

```
    user = forms.CharField(max_length = 100)
```

```
    password = forms.CharField(widget = forms.PasswordInput())
```

As seen above, the field type can take "widget" argument for html rendering; in our case, we want the password to be hidden, not displayed. Many others widget are present in Django: DateInput for dates, CheckboxInput for checkboxes, etc.

Using Form in a View

There are two kinds of HTTP requests, GET and POST. In Django, the request object passed as parameter to your view has an attribute called "method" where the type of the request is set, and all data passed via POST can be accessed via the request.POST dictionary.

Let's create a login view in our myapp/views.py –

```
#-*- coding: utf-8 -*-

from myapp.forms import LoginForm

def login(request):

    username = "not logged in"

    if request.method == "POST":

        #Get the posted form

        MyLoginForm = LoginForm(request.POST)

        if MyLoginForm.is_valid():

            username = MyLoginForm.cleaned_data['username']

        else:

            MyLoginForm = Loginform()

    return render(request, 'loggedin.html', {"username" : username})
```

The view will display the result of the login form posted through the loggedin.html. To test it, we will first need the login form template. Let's call it login.html.

<html>

```
<body>

<form name = "form" action = "{ % url "myapp.views.login" % }"

    method = "POST" >{ % csrf_token % }

    <div style = "max-width:470px;">

        <center>

            <input type = "text" style = "margin-left:20%;"

                placeholder = "Identifiant" name = "username" />

        </center>

    </div>

    <br>

    <div style = "max-width:470px;">

        <center>

            <input type = "password" style = "margin-left:20%;"

                placeholder = "password" name = "password" />

        </center>

    </div>

    <br>

    <div style = "max-width:470px;">

        <center>

            <button style = "border:0px; background-color:#4285F4; margin-top:8%;

                height:35px; width:80%;margin-left:19%;" type = "submit"
```

```
        value = "Login" >

        <strong>Login</strong>

    </button>

</center>

</div>

</form>

</body>

</html>
```

The template will display a login form and post the result to our login view above. You have probably noticed the tag in the template, which is just to prevent Cross-site Request Forgery (CSRF) attack on your site.

```
{% csrf_token %}
```

Once we have the login template, we need the loggedin.html template that will be rendered after form treatment.

```
<html>

<body>

    You are : <strong>{{ username }}</strong>

</body>

</html>
```

Now, we just need our pair of URLs to get started: myapp/urls.py

```
from django.conf.urls import patterns, url

from django.views.generic import TemplateView
```

```
urlpatterns = patterns('myapp.views',  
  
    url(r'^connection/', TemplateView.as_view(template_name = 'login.html')),  
  
    url(r'^login/', 'login', name = 'login'))
```

When accessing `"/myapp/connection"`, we will get the following `login.html` template rendered –

Setting Up Sessions

In Django, enabling session is done in your project `settings.py`, by adding some lines to the `MIDDLEWARE_CLASSES` and the `INSTALLED_APPS` options. This should be done while creating the project, but it's always good to know, so `MIDDLEWARE_CLASSES` should have –

```
'django.contrib.sessions.middleware.SessionMiddleware'
```

And `INSTALLED_APPS` should have –

```
'django.contrib.sessions'
```

By default, Django saves session information in database (`django_session` table or collection), but you can configure the engine to store information using other ways like: in file or in cache.

When session is enabled, every request (first argument of any view in Django) has a `session` (dict) attribute.

Let's create a simple sample to see how to create and save sessions. We have built a simple login system before (see Django form processing chapter and Django Cookies Handling chapter). Let us save the username in a cookie so, if not signed out, when accessing our login page you won't see the login form. Basically, let's make our login system we used in Django Cookies handling more secure, by saving cookies server side.

For this, first let's change our login view to save our username cookie server side –

```
def login(request):  
  
    username = 'not logged in'  
  
    if request.method == 'POST':
```

```
MyLoginForm = LoginForm(request.POST)

if MyLoginForm.is_valid():

    username = MyLoginForm.cleaned_data['username']

    request.session['username'] = username

else:

    MyLoginForm = LoginForm()

return render(request, 'loggedin.html', {"username" : username})
```

Then let us create formView view for the login form, where we won't display the form if cookie is set –

```
def formView(request):

    if request.session.has_key('username'):

        username = request.session['username']

        return render(request, 'loggedin.html', {"username" : username})

    else:

        return render(request, 'login.html', { })
```

Now let us change the url.py file to change the url so it pairs with our new view –

```
from django.conf.urls import patterns, url

from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',

    url(r'^connection/', 'formView', name = 'loginform'),

    url(r'^login/', 'login', name = 'login'))
```

When accessing /myapp/connection, you will get to see the following pag

CHAPTER-07

SYSTEM TESTING

7.1 System Test

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.2 Types Of Tests

7.2.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

7.2.2 Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

7.2.3 Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

7.2.4 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

7.2.5 White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

7.2.6 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box, you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

7.2.7 Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

7.2.8 Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

7.2.9 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

7.2.10 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

7.3 Test Cases

Test Case No.	Test Scenario	Test Steps	Expected Outcome
1	Valid Input Data	1. Input valid data for all fields (e.g., FL_DATE, TAIL_NUM, CRS_DEP_TIME).	Form should submit successfully and display prediction result.
		2. Submit the form.	
2	Invalid Flight Date	1. Input an invalid or future date for FL_DATE.	The form should reject submission and show an error message like "Invalid date."
		2. Submit the form.	
3	Missing Required Fields	1. Leave some required fields (e.g., OP_UNIQUE_CARRIER) blank.	Error messages should indicate missing fields, and no prediction is made.
		2. Submit the form.	
4	Predicting for Canceled Flight	1. Set CANCELLED to 1.	The system should display a message like "Flight is canceled," and no prediction is made.
		2. Fill other fields with valid data.	
		3. Submit the form.	
5	Invalid Airport ID	1. Input invalid data for ORIGIN_AIRPORT_ID or DEST_AIRPORT_ID.	Error message should display "Invalid Airport ID."
		2. Submit the form.	
6	Invalid Flight Number Format	1. Input an invalid (e.g., alphanumeric) flight number in OP_CARRIER_FL_NUM.	The form should reject submission and show an error message.
		2. Submit the form.	

7	Future Flight Prediction	1. Input a future flight date.	The prediction should run for the future flight, and a result should be displayed.
		2. Fill in valid data.	
		3. Submit the form.	
8	Correct Handling of Optional Fields	1. Leave optional fields (e.g., DEP_TIME) empty.	Form should submit successfully, and a prediction should be made based on available data.
		2. Submit the form.	
9	Handling Large Inputs	1. Input large values for fields like DISTANCE or TAXI_OUT.	The form should accept the inputs, and a prediction should be made without errors.
		2. Submit the form.	

CHAPTER-08

RESULTS

Home Page

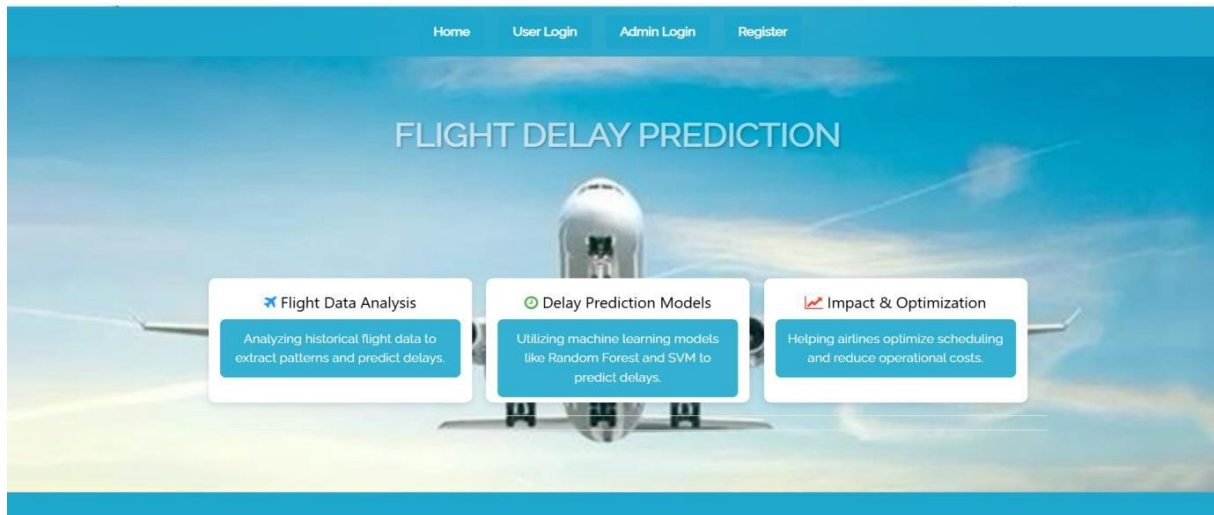


Fig 8.1 Flight Delay Forecasting home page

User register Form

The screenshot shows the user registration form on the same website. The form is titled "SIGN UP" and includes a sub-header "Please fill in this form to create an account." The form fields are: Username (with a placeholder "Enter Username"), Email (with a placeholder "Enter Email"), Phone Number (with a placeholder "Enter Phone Number"), Password (with a placeholder "Enter Password"), Address (with a placeholder "Enter Address"), and Status (with a dropdown menu showing "waiting"). A green "Sign Up" button is at the bottom of the form.

Fig 8.2 Flight Delay Forecasting User Registration Form Page

User Login Page

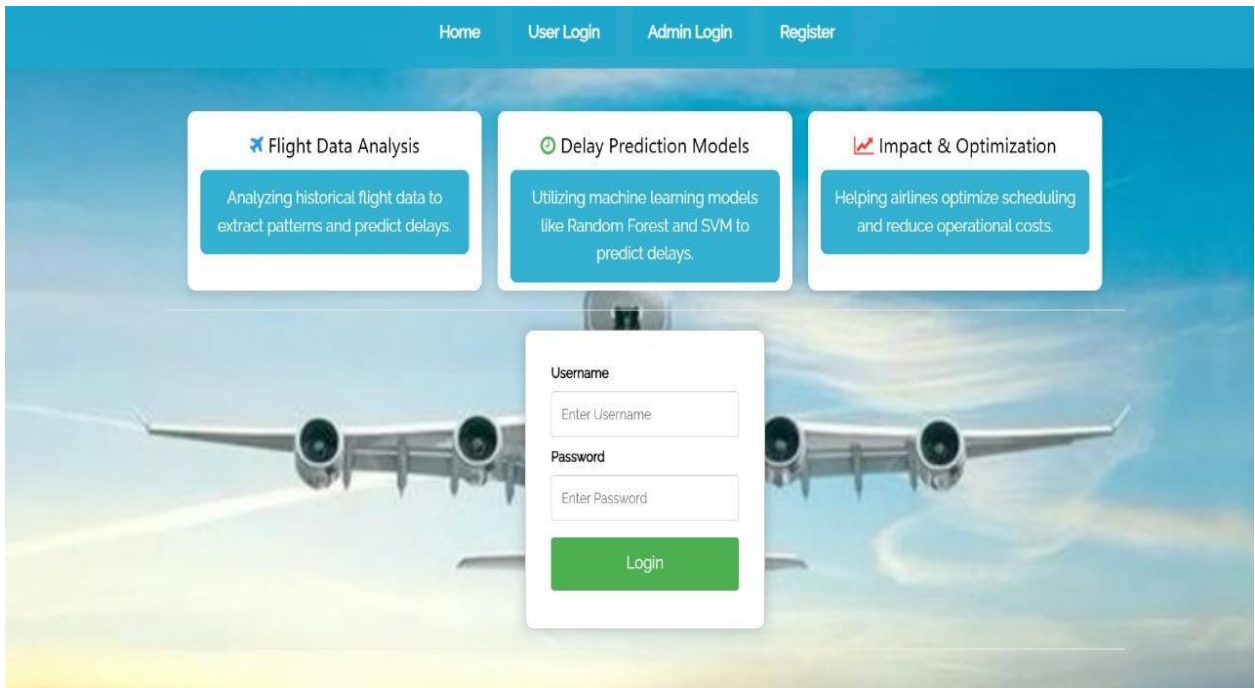


Fig 8.3 Flight Delay Forecasting User Login Page

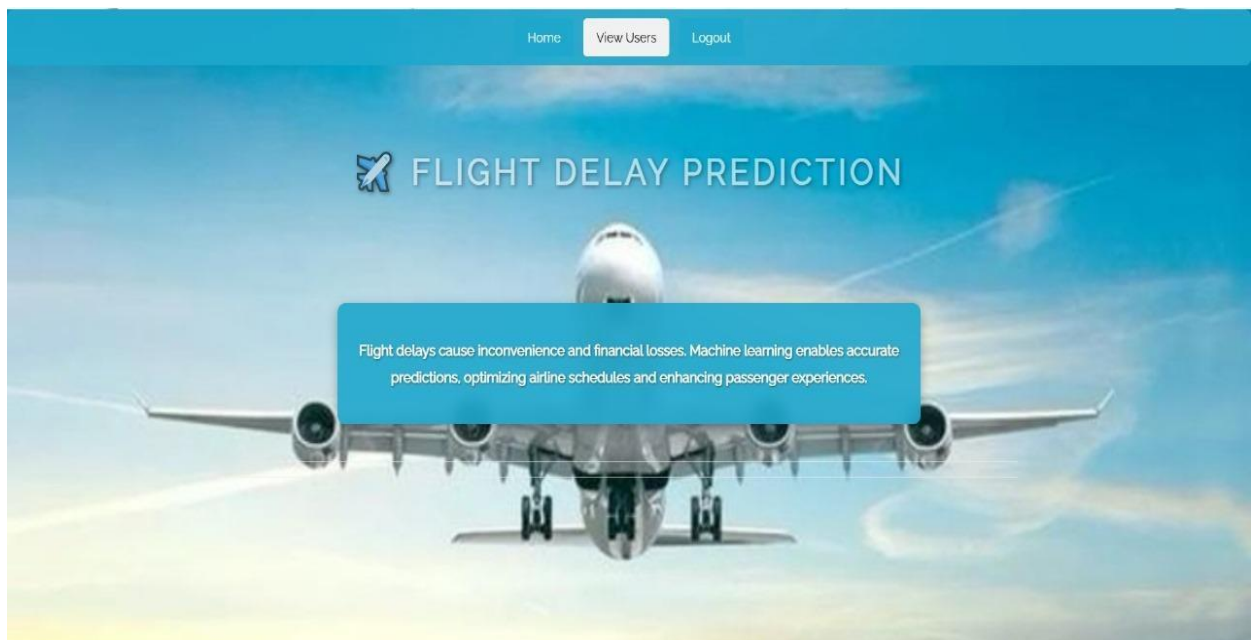


Fig 8.4 Flight Delay Forecasting After Admin Login Page

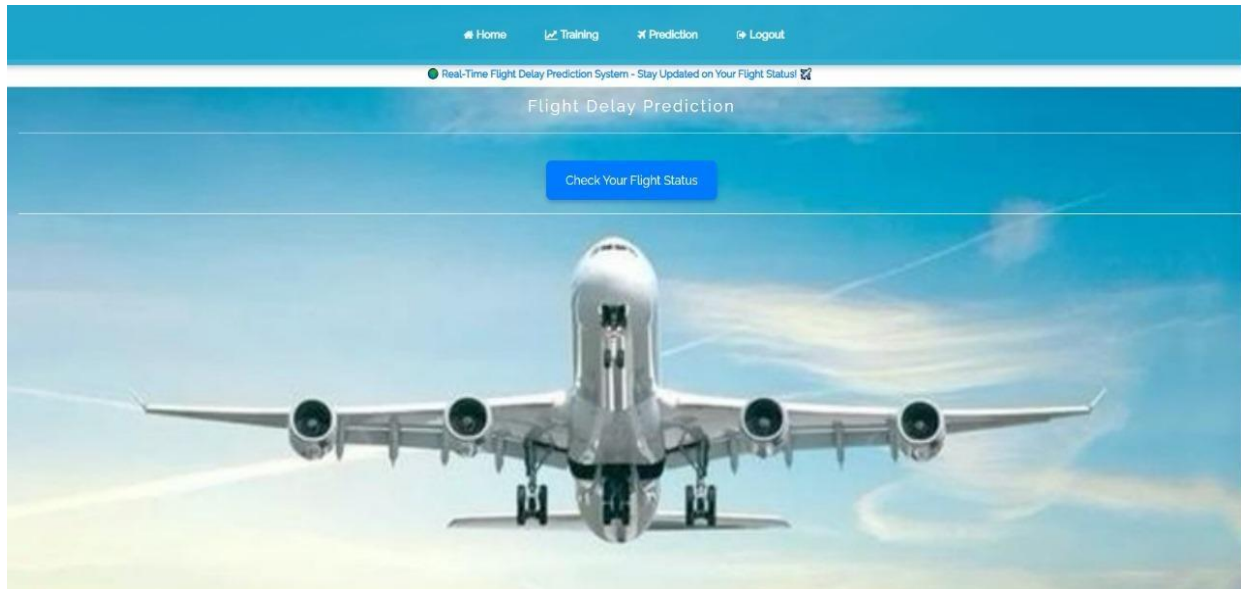


Fig 8.5 Flight Delay Forecasting page

The screenshot displays the input form for the "Flight Delay Prediction" system. The form is overlaid on the same airplane background as the previous page. It contains the following fields and controls:

- Flight Date:** A text input field with a placeholder "mm/dd/yyyy" and a calendar icon.
- Unique Carrier:** A text input field.
- Operating Carrier:** A text input field.
- Tail Number:** A text input field.
- Flight Number:** A text input field.
- Origin Airport ID:** A text input field.
- Origin Airport:** A text input field.
- Destination Airport ID:** A text input field.
- Destination Airport:** A text input field.
- Scheduled Departure Time:** A text input field with a clock icon.
- Actual Departure Time:** A text input field with a clock icon.
- Taxi Out Time:** A text input field.
- Flight Distance:** A text input field.

A blue "Predict" button is located at the bottom of the form.

Fig 8.6 Flight Delay Forecasting Input Form

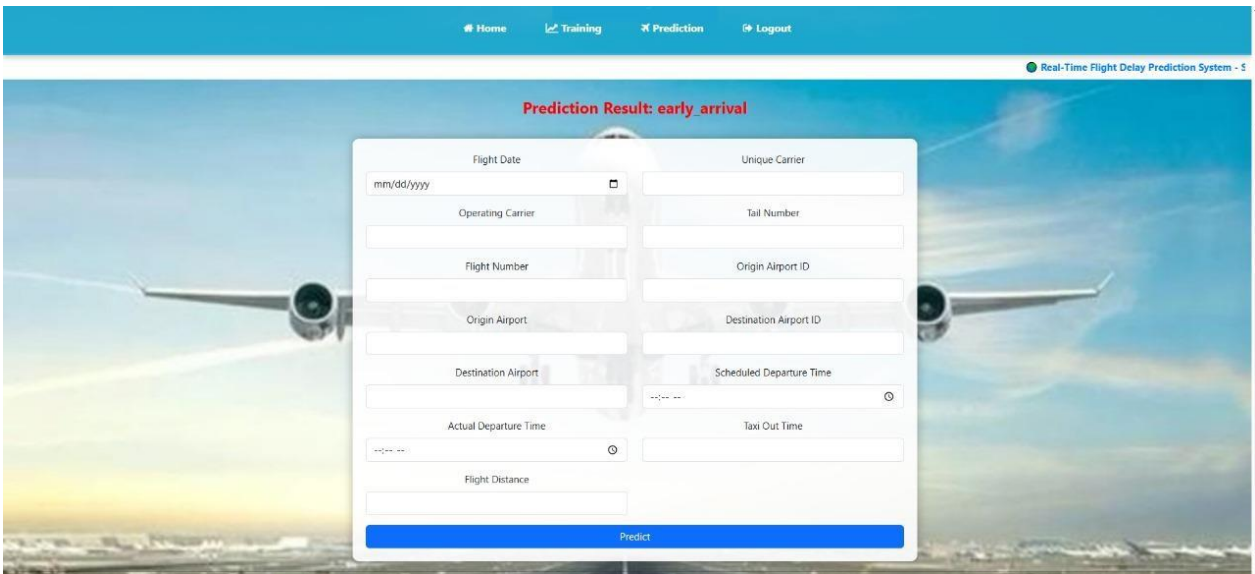


Fig 8.7 Flight Delay Forecasting result

User Side graphs

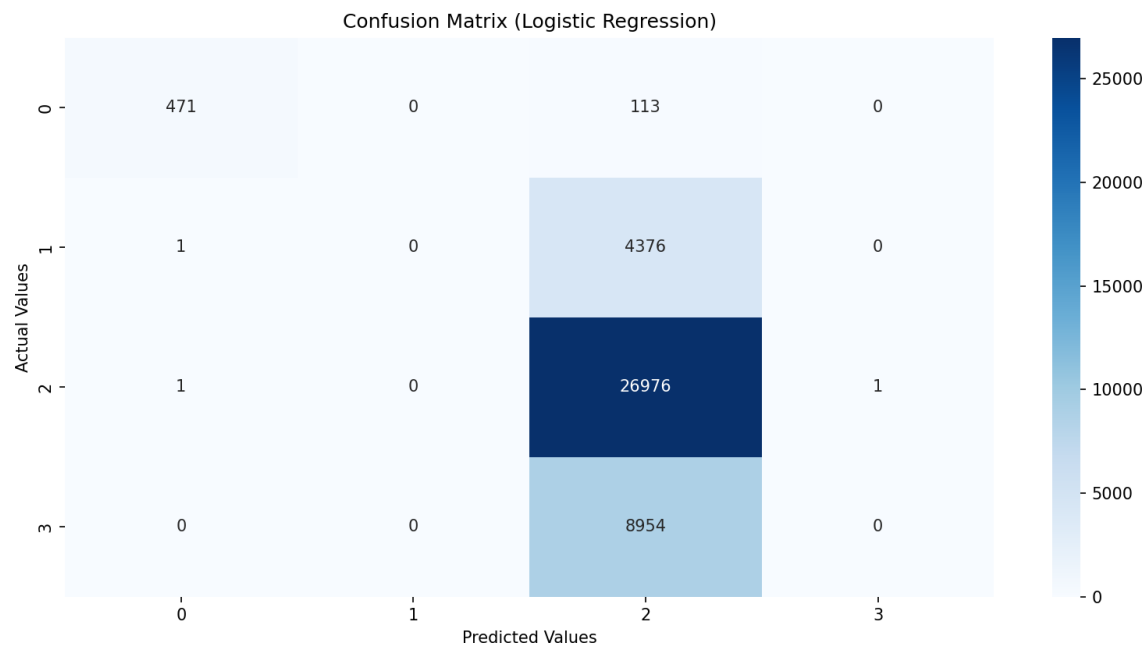


Fig 8.8 Confusion Matrix for Logistic Regression

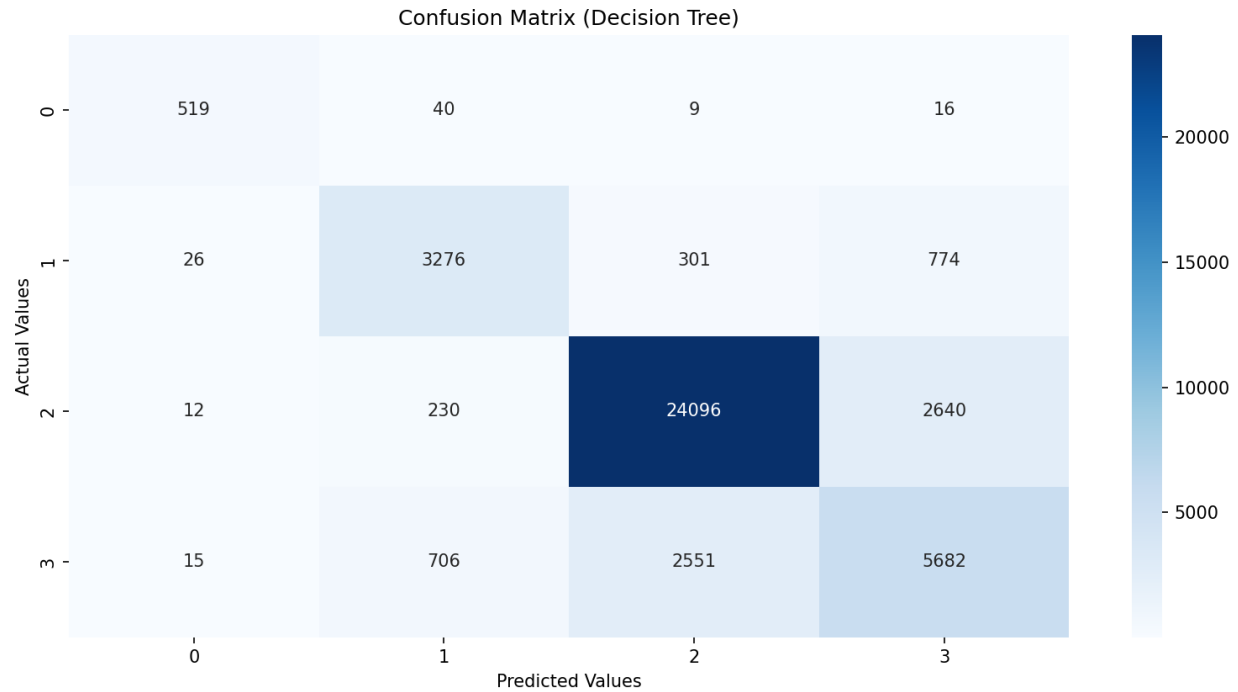


Fig 8.9 Confusion Matrix for Decision Tree

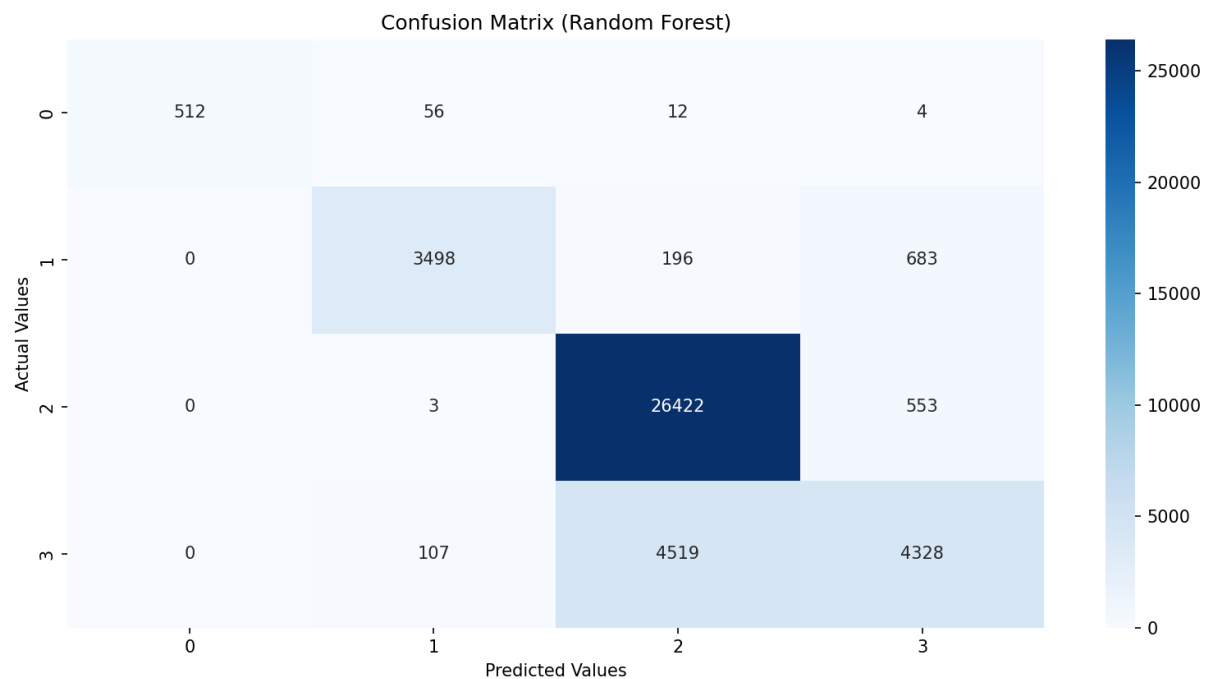


Fig 8.10 Confusion Matrix for Random Forest

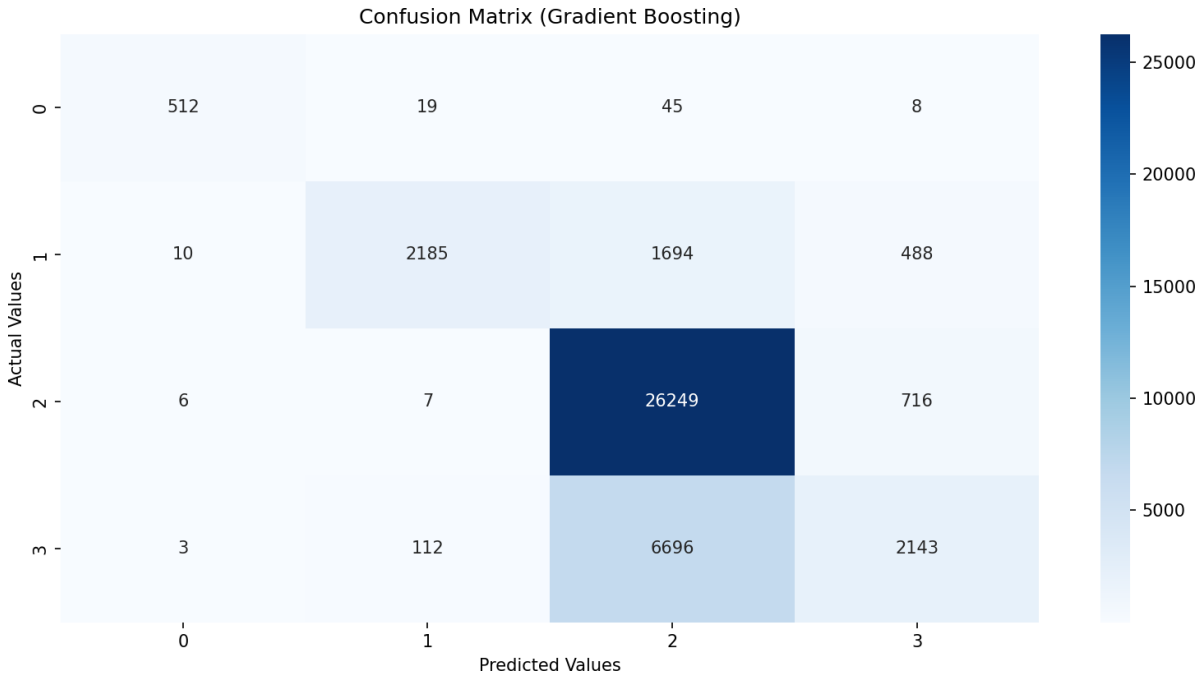


Fig 8.11 Confusion Matrix for Gradient Boosting

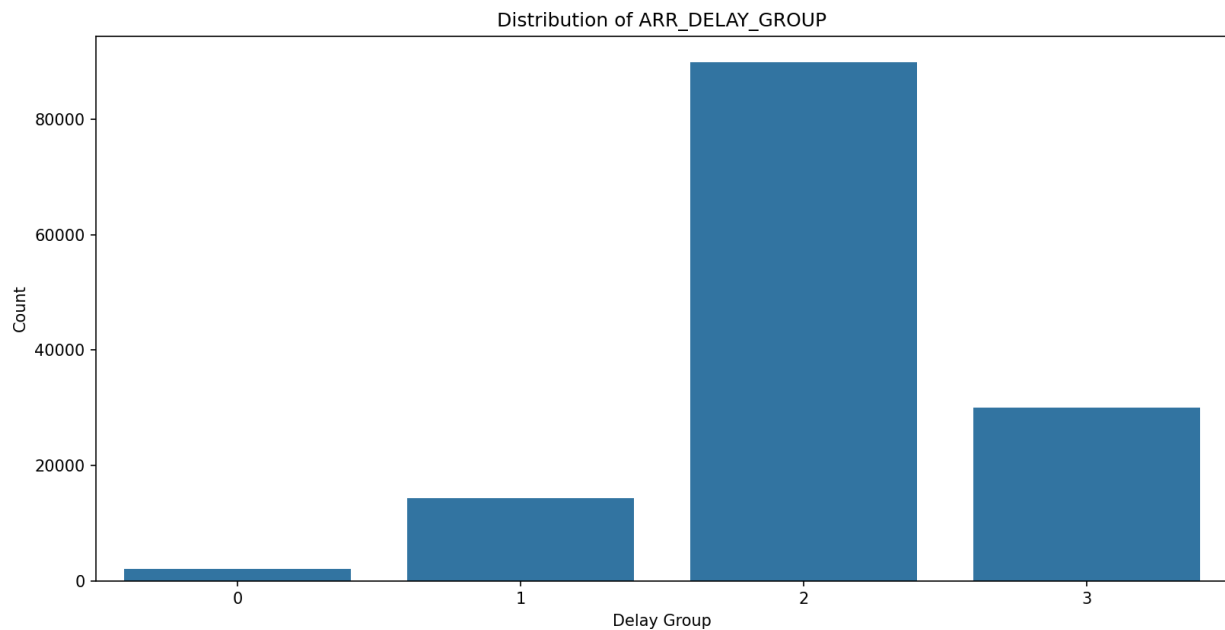


Fig 8.12 Distribution of Flight Delay Groups

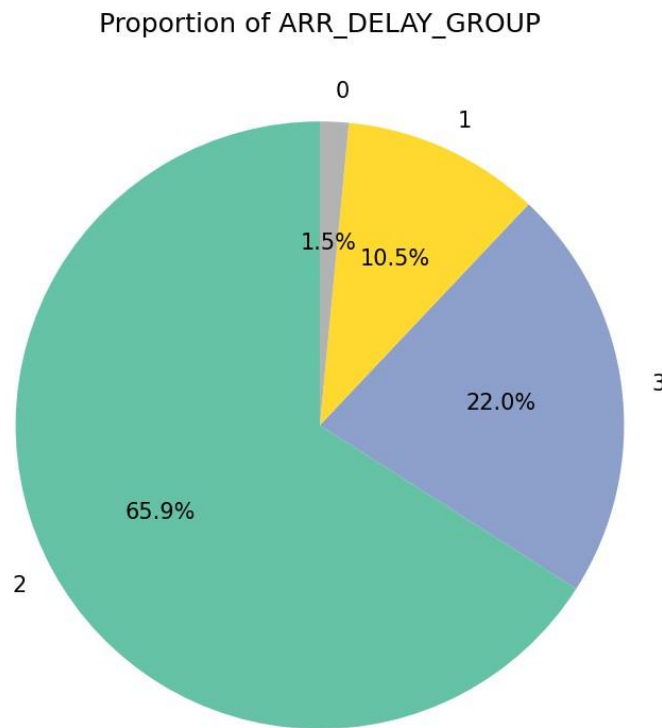


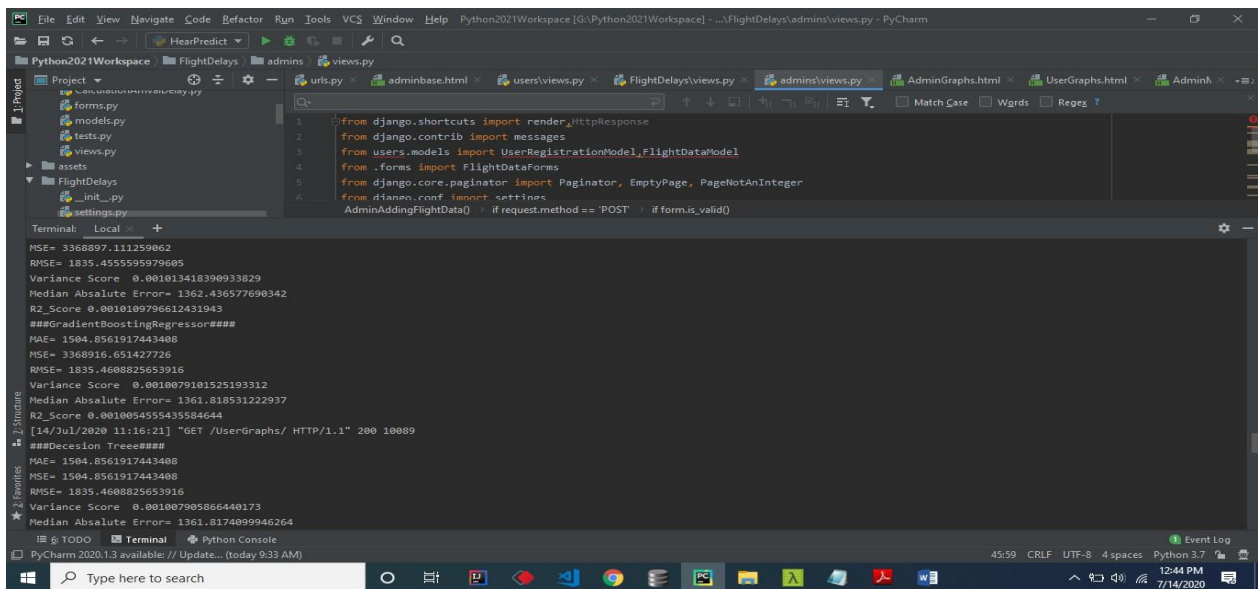
Fig 8.13 Proportion of Flight Delay Groups

Model Performance

TRAINING RESULTS	
Model	Accuracy (%)
Logistic Regression	67.12
Decision Tree	82.23
Random Forest	85.44
Gradient Boosting	76.02

Fig 8.14 Model Performance Comparison on Training Data

Server side results



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help Python2021Workspace [G:\Python2021Workspace] - ...FlightDelays\admins\views.py - PyCharm
Python2021Workspace FlightDelays admins admins\views.py
Project
  forms.py
  models.py
  tests.py
  views.py
  FlightDelays
    __init__.py
    settings.py
  urls.py
  adminbase.html
  users\views.py
  FlightDelays\views.py
  admins\views.py
  AdminGraphs.html
  UserGraphs.html
  AdminA...
  Match Case Words Regex
1 from django.shortcuts import render,HttpResponse
2 from django.contrib import messages
3 from users.models import UserRegistrationModel,FlightDataModel
4 from .forms import FlightDataForm
5 from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
6 from django.conf import settings
  AdminAddingFlightData() if request.method == 'POST' if form.is_valid()

Terminal Local
MSE= 3368897.111259862
RMSE= 1835.4555595979695
Variance Score 0.001013418390913829
Median Absolute Error= 1362.436577690342
R2_Score 0.0010109796612431943
###GradientBoostingRegressor###
MAE= 1504.8561917443408
MSE= 3368916.651427726
RMSE= 1835.4608825653916
Variance Score 0.0010079101525193312
Median Absolute Error= 1361.818531222937
R2_Score 0.0010054555435584644
[14/Jul/2020 11:16:21] "GET /UserGraphs/ HTTP/1.1" 200 10089
###Decision Tree###
MAE= 1504.8561917443408
MSE= 1504.8561917443408
RMSE= 1835.4608825653916
Variance Score 0.001007905866440173
Median Absolute Error= 1361.8174099946264
IE TODO Terminal Python Console
PyCharm 2020.1.3 available // Update... (today 9:33 AM)
45:59 CRLF UTF-8 4 spaces Python 3.7
7/14/2020
```

Fig 8.15 Flight Delay Fore casting Server Side Results

CHAPTER-09

CONCLUSION

Our research showed that aircraft delay predictions may be made with good accuracy using machine learning methods. In addition to evaluating delays for different human requirements, the previously mentioned categorization and analysis aims to closely examine elements that impact delays, including ‘weather delay,’ ‘airline delay,’ and ‘security delay.’ The Gradient Boosting model performed the best, correctly categorizing delays into two groups using the previously mentioned parameters including ‘Departure Time,’ ‘Air Time,’ and ‘Month’—with 100% accuracy. Thus, it can be effectively used to predict flight delays, which will be beneficial for airports and airlines, as well as passengers. Therefore, the study of flight delays presented in this paper is grounded entirely in scientific parameters, underscoring its crucial significance in the aviation industry.

CHAPTER-10

FURTHER ENHANCEMENT

10.1. Explore Additional Machine Learning Algorithms

- The paper you uploaded also explores CNN-LSTM deep learning frameworks for improving delay predictions. These models capture sequential data better, which could be helpful if you have time-series data or patterns (Flight_Delay_Prediction...).

10.2. Feature Engineering

- **Time-based features:** Incorporate features such as the day of the week, month, or time of the day to improve the accuracy of the predictions. Flight delay data often has time-based patterns.
- **Weather Data:** Integrate weather information, which is often a strong predictor of delays, as suggested by various studies (Flight_Delay_Prediction...).
- **Handling Dates:** Consider deriving features like the time difference between START_DATE and END_DATE, time of day (morning/afternoon/evening), and seasonal indicators.

10.3 Use of Deep Learning Models

- We do experiment with deep learning models like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), particularly LSTMs, as mentioned in the uploaded document for handling sequential data (Flight_Delay_Prediction...).

CHAPTER-11

REFERENCES

- [1] Borse, Y., Jain, D., Sharma, S., Vora, V., and Zaveri, A. (2020) Flight Delay Prediction System. International Journal Of Engineering Research & Technology (IJERT) Volume 09, Issue 03 (March 2020).
- [2] Bureau of Transportation Statistics. Available online: <https://www.bts.gov/> (accessed on 26 March 2020).
- [3] Akpınar, M.T. and Karabacak, M.E. (2017). Data mining applications in civil aviation sector: State-of-art review. In CEUR Workshop Proc (Vol. 1852, pp. 18-25).
- [4] Nazeri, Z. and Zhang, J. (2017). Mining Aviation Data to Understand Impacts of Severe Weather. In Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC.02) pp. 518-523.
- [5] Ha. ., J. N. a. H. P. S. Man. (2015) "Analysis of Air-Moving on Schedule Big Data based on CrispDm Methodology," ARPN Journal of Engineering and Applied Sciences, pp. 2088-2091.
- [6] Mukherjee, A., Grabbe, S. R., and Sridhar, B. (2014). Predicting Ground Delay Program at an airport based on meteorological conditions. In 14th AIAA Aviation Technology, Integration, and Operations Conference (pp. 2713-2718).
- [7] Natarajan, V., Meenakshisundaram, S., Balasubramanian, G. and Sinha, S. (2018) "A Novel Approach: Airline Delay Prediction Using Machine Learning," 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA pp. 1081-1086, doi: 10.1109/CSCI46756.2018.00210
- [8] Tu, Y., Ball, M.O, and Jank, W.S. (2008) Estimating flight departure delay distributions—a statistical approach with long-term trend and short-term pattern", Journal of the American Statistical Association, 103, pp. 112 125.
- [9] Mueller, E.R. and Chatterji, G.B. (2002) Analysis of aircraft arrival and departure delay characteristics", In AIAA's Aircraft Technology, Integration, and Operations (ATIO) 2002 Technical Forum, Los Angeles, California,

CHAPTER-12

PAPER PUBLICATION CERTIFICATIONS



VELAMMAL
INSTITUTE OF TECHNOLOGY

Chennai - Kolkatta Highway, Panchetti, Ponneri

**5th International Conference on Artificial Intelligence,
6G Communications and Network Technologies - ICA6NT 2025**

Certificate of Appreciation

This is to certify that

DEVANGAM HARATHI

Anantha Lakshmi Institute Of Technology And Sciences

has participated and presented paper entitled

Machine Learning Techniques For Accurate Flight Delay Forecasting

at the 5th International Conference on Artificial Intelligence, 6G Communications and Network Technologies (ICA6NT 2025) organized by the Department of Electronics and Communication Engineering, Velammal Institute of Technology, Chennai held on 27th & 28th , March 2025.


 Coordinator
 Dr. R. Jothi Chitra
 Professor-ECE


 Coordinator
 Dr. M. Sivarathinabala
 Professor-ECE


 Coordinator
 Mr. K. Ragupathi
 Assistant Professor-ECE


 Convener
 Dr. B. Sridevi
 Professor & Head -ECE


 Conference Chair
 Dr. N. Balaji
 Principal











